Universitetet i Oslo
Institutt for informatikk

Teaching object oriented programming to youths using the control technology Lego Mindstorms.

Roar Granerud
Department of Informatics
University of Oslo

February 2005

Preface

Abstract

This master thesis is a culmination of two experiments performed with children. The children got to play with Lego Mindstorms, build and program their custom made Lego robots. The first of the experiments was performed with 11 year olds that used Lego Robolab when programming their Lego robots. This was a pilot experiment prior to the second experiment where 14 year olds programmed the same Lego Mindstorms robot, but this time using a custom made Java API called Lejos (*LEJOS. Java for the RCX* n.d.).

This experiment was the last of many experiments performed by people from the Comprehensive Object Oriented Learning - project (COOL). One of the aims of COOL is exploring the complex area of learning and teaching object oriented concepts. The point of the study described in this thesis was to see in what way the control technology Lego Mindstorms could be used to teach children object oriented concepts. Control technology enables devices to be programmed to achieve goals. It can be something as simple as a tape recorder.

In addition to the results from the experiments performed, this thesis describes the tools used to program the Lego Mindstorms robots and gives an indication of how good these tools were both for learning programming in general and learning object oriented concepts.

The results from these experiments show that children from 11 to 15 years find Lego Mindstorms fun, but it is very difficult to use and takes a lot of energy, so they get tired after three days. Girls is less fascinated by Lego Mindstorms than boys, but is still able to learn just as much as the boys.

Lego Robolab cannot be used as a tool for teaching object oriented concepts. But the Java API Lejos enabled the children to learn a lot about the object oriented concept of encapsulation. The children got a good sense of the program flow of their programs when programming physical Lego robots that did or did not do what they were told to do. Using control technology is still very time consuming, and there are some problems involved with using physical Lego robots compared to virtual robots on the computer screen.

Acknowledgments

I would like to thank my supervisors Christian Holmboe and Jens Kaasbøll for their guidance through the work on this thesis. Additionally I want to thank the rest of the COOL group, especially Ole Smørdal, Annita Fjuk and Richard E. Borge whom I worked very closely with during the experiments. And I also want to thank Arne-Kristian Groven for some helpful hints along the way. The COOL groups is a good bunch that I am proud to say that I have worked with.

Contents

List of Figures

Chapter 1

Introduction

1.1 Focus areas in this thesis

1.1.1 Children programming

This thesis will present some results on how the children cooperate when trying to solve a difficult assignment. How boys and girls at age 11 and age 14 differ from each other in the way they work with the problems presented to them. I will try to find out how the children are thinking about the computer program they are writing compared to how the physical robot works in the real world.

1.1.2 Lego Mindstorms and object oriented programming

The point of the experiments described within was using control technology as a tool for teaching object oriented concepts. In this case the control technology used was Lego Mindstorms. We wanted to find out that if the objects inside the computer program is represented by physical objects, would that be helpful when building a cognitive understanding of the object oriented Java program structure? The main object oriented concept I want to focus on in this thesis is the object oriented concept encapsulation. See chapter 2.2.1 on page 13 for a description of encapsulation.

We used an objects first approach to programming during the experiments. This means that we introduced objects and object oriented structuring right from the start, with a complex start program, instead of starting with a less complex "hello world" program. See chapter 2.4 on page 18 for a more detailed description of the objects first approach. (Borge & Kaasbøll 2003) say that almost all computer science courses starts with this kind of program, as an introduction to computer programming.

1.1.3 Tools

Finally I will try to describe how the tools for programming the Lego Mindstorms (*Lego Mindstorms* n.d.) robots were used, and what problems occurred during the assignments. Some of the tools were made by Lego for programming their Lego Mindstorms robots. Lego's programming tool was called Robolab (*Lego Mindstorms for schools* n.d.). But the tools from Lego did not suffice when teaching object oriented concepts. So we used a third party made Java API called Lejos (*LEJOS. Java for the RCX* n.d.), customized to our needs. How good these tools are, will also be presented in this thesis.

1.2 Readers' Guide

This thesis covers two experiments using Lego. The first experiment was at elementary school Fjellhamar. This experiment was a pilot experiment to find out what we could do with the Lego tool. See chapter 4.4 on page 44 for a detailed description of the experiment. We also wanted to test the technical equipment. This equipment consisted of the physical Lego bricks and the cameras we used to videotape the whole event. Therefore this first experiment is not covered as deep in this thesis as the second experiment is. During the process of writing this thesis, I used less time looking on the video taped material from this first experiment than I did with the last.

When I started my master study I wanted to find out how Lego Mindstorms could be used to teach object oriented concepts, and what concepts could be thought. There was little focus on these concepts at Fjellhamar, and this is also a reason why I have focused on the results from the other experiment, the one at Mølladammen. See chapter 4.5 on page 46 for a detailed description of the experiment at Mølladammen.

During the process of writing this masters thesis I attended a course at the University of Oslo called Informatics Didactics. During this course I and two other students wrote a small article about the experiment that took place at Mølladammen. This article is part of the appendix of this thesis. I have used some of the text from this article in my thesis since it was very relevant and I was the main author of the article as well.

In this thesis I first write some theory about teaching and learning in general and in the field of object orientation. See chapter 2.1 on page 12. Then in chapter 3.1 on page 21 i describe the different tools used in the experiments. Many of these tools are new in this field, and therefore I have dedicated a lot of pages describing their use. Then I describe, in chapter 4.1 on page 40, the methods used. At the end in chapter 5 on page 57, I present the different results under different sub-chapters. Chapter 6 on page 92 is the last chapter and is a summary of all the results presented in this thesis. In this last chapter I also try to draw a conclusion from the results.

1.3 A story from my own experience

I played a little with Pascal programming before I started my studies at the University of Oslo. So when I attended CS1 I felt that this should be a piece of cake and not pose much of a challenge. The study started with simple procedural programming, learning basic Java syntax and how to use loops and if-statements. Then half way through the course, object oriented concepts were introduced, and we were supposed to rewrite some of the programs we had already written. This time in an object oriented fashion. I did not understand anything of what we were supposed to do, but I managed to "solve" all the tasks I was given, so I did not bother to understand what object orientation was all about. Then the exam came up and I thought it was very easy, solved all the programming assignments and felt that this was a good exam. Then the result came and I found out that I failed the course. And all my friends that had a similar experience in programming either failed or got a very bad grade.

This is a typical problem when changing from procedural programming to object oriented programming. Stubborn people, like me, do not understand or do not want to understand what all the fuss with object oriented programming is all about. What difference does it do when the program works the way it should? During the summer vacation after receiving bad news about my CS1 class I suddenly understood what it was all about, and had the exam again and had pretty good results.

This is the reason why I wanted to write a thesis about a way to learn Object Oriented programming at an early stage, even before learning anything else about both programming and programming paradigms.

1.4 Disclaimers

I would like to comment my use of the words Lego and Lego Mindstorms during this thesis. I should have written Lego® and Mindstorms ™ during this thesis, but I use those words a lot and I find that the ® and ™ symbols are a disturbing element when reading a text. So therefore I left them out. The same goes for Robolab™ , JCreator™ , Java™and Sun Microsystems™.

One last thing is that the experiments described withing was performed at one 6th grade class and one 9th grade class. In Norway children are usually 11 to 12 years old when attending 6th grade and 14-15 years old when attending 9th grade. But I only use the number 11 and 14 during this thesis, so it would be easier to write and read.

Chapter 2

Background

In this chapter I will briefly explain what the object oriented paradigm is all about and what the COOL group is doing. I will also mention some well known theories for learning, and finally I want to discuss learning to program in an object oriented way in your first programming course instead of using the more traditional procedures first approach.

2.1 The COOL project

This is a masters thesis written at the end of a project started in 2002. This project is called COOL, which is short for Comprehensive Object Oriented Learning (*COOL* 2002). The experiments described within this thesis was the last experiments of many COOL experiments. Me as a master student only participated in these last experiments and the analyzing of the results. Some of the results found during the writing of this thesis are also described in the part of the COOL anthology written by Professor Jens Kaasbøll and me. It can be found as an attachment to this thesis. This anthology is a summary of the COOL project.

The COOL project was started by Kristen Nygaard in 2002 and is a project with many participants from different academic environments. According to Nygaard, the main objectives of the COOL project are:

> Exploring the complex area of learning (and teaching) object oriented concepts; Maintaining and further developing the Norwegian (Scandinavian) heritage from object-orientation; Designing blended learning environments.

Most pedagogical approaches state that in order to learn something, the example has to be sufficiently simple. The examples are from earlier, traditional programming. (Groven, Hegna & Smørdal 2003) has a description of the Scandinavian heritage from object orientation where Kristen Nygaard says that we need a sufficiently complex example that introduces the basic object oriented concepts from the very beginning.

2.2 Object Oriented programming

The objects in object oriented programming are instances of classes. When learning object oriented programming myself in my CS1 course at the university, the teacher said that a class was the blueprint of a house, while the object was the actual house based on that blueprint. The blueprint defined that we needed something with walls, windows and at least one door. Then when you made an object of this class, you made a house with walls, windows and doors, and you decided that the doors should be blue and the walls should be green. This analogy did not work for me. I did not understand what object oriented programming was all about until after my exam. As mentioned in chapter 2.1 on the page before the COOL projects tries to find other ways to teach object oriented programming.

There are lots of object oriented concepts. In this thesis I only want to mention the concept that we wanted to teach during the experiments mentioned in this thesis. This is the concept of encapsulation, which I feel is one of the main object oriented concept.

2.2.1 The object oriented concept Encapsulation

Three terms that are used a lot in object oriented design are abstraction, encapsulation and information hiding. (Berard 2000) discusses different dictionary definitions and his conclusion is:

> Abstraction, information hiding, and encapsulation are very different, but highly-related, concepts. One could argue that abstraction is a technique that helps us identify which specific information should be visible, and which information should be hidden. Encapsulation is then the technique for packaging the information in such a way as to hide what should be hidden, and make visible what is intended to be visible.

I do not emphasize much on the hiding part of encapsulation in this thesis, but rather putting the data where it “belongs”. The belong-term is very vague, but what I mean is that information about a house should be located inside the house in the same way that a person knows its age, name and so on.

Encapsulation means that the different objects in a computer program has control over its own data and changes it with its own methods. This information is often not available outside the object, so in order to access the information, you have to use the objects own methods to return the value you want. The only thing you got outside the object is a reference to the object. The following example has a reference to a Person object called *person1* :

If you create a person object, this object knows everything there is to know about itself. If you want to know the name of this person, you ask the person what name it has. In Java this could be done by the call
person1.getName()
In this example the object got a method getName() that returns its own name to the one wanting to know it. If you wanted to change the name of this object, you might have to use a method like:
person1.setName("Svein")
You send the name you wanted the person object to have as a parameter, and the person object fixes the rest by itself. The setName method probably updates this objects data, so that when someone uses the getName method in the future, the name Svein is returned.

An idea behind the encapsulation concept is sharing the responsibility. The different objects in an object oriented program has their own responsibilities.

2.2.2 Learning object orientation

Détienne writes in her book Software Design - Cognitive Aspects(Detienne 2002, page 58):

> The identification of objects ought to be easy since the objects form a natural representation of the problem entities. According to (Meyer 1988) the world is naturally structured into objects. It therefore seems particularly appropriate to organize solutions to design problems around the programming representations of these objects. The mapping between the problem domain and the solution domain ought to be simple. The objects of the problem domain are identified and then used to structure the OO system. Thus both the problem and the solution are decomposed on the basis of objects.

Programming something is easier when the programmer can see the results from his or her programming. It is easier to find the bugs when you can see just where in a program the error is, because the thing you are programming is in direct correspondence with your programming code. In this thesis we used a real world Lego robot with physical motors and sensors. When programming this kind of robot the programmer would want to give the desired command to the object in the program that represented the real world object. I.e, when you want to make a vehicle with wheels go forward, you tell the motors that run the wheels to go forward. This is very easily transferred from the real world into an object oriented program.

Still, as (Detienne 2002, page 59) says:

> It is worth remarking that early books on OO emphasized

how easy it was to identify objects while later ones, often by the same authors, emphasize the difficulty of identifying them.

But when the objects already are identified, like in the programming exercise described in this thesis, this difficulty does not apply.

2.3 Learning theory

There are three major theories for how people learn. I will try to explain in short what the different theories are about. The theories are:

1. Behaviorism - Acquiring and applying associations: The behaviorist / empiricist view.
2. Constructivism - Creating and using conceptual and cognitive structures: The cognitive / rationalist view.
3. Situated cognition - Becoming attuned to constraints and affordances through participation: The situative / pragmatist-sociohistoric view.

These theories are well explained in the article by (Greeno & Collins 1996).

2.3.1 Behaviorism

Knowledge is empirical. An animal or human that does not show any change in behavior has not learned anything. Knowledge is acquired through positive or negative reinforcement. Learning is to strengthen or adjust the associations between ideas or stimuli and responses. (Borge 2004) says that behaviorism bases itself on that knowledge is something that is out there in the world or something that is inside people. And the way to get to this knowledge is through the correct kind of stimulus and reinforcement.

When training horses and other animals you very often use positive or negative reinforcement. In the following example I want a horse to move forward every time i sound a whistle. If you sound a whistle and at the same time push the horse forward, the horse will either stand its ground or move forward. If the horse moves forward, you give it a piece of sugar. This is positive reinforcement. The horse will remember that when it heard a whistle, was pushed forward, and took a step forward, it received a piece of sugar. On the other hand, if the horse stood still instead of moving forward, he would taste the whip. This is negative reinforcement. The horse will associate the sound of the whistle, the pushing and standing still with the whipping. Hopefully next time the whistle is blown and the horse is pushed forward it would want to receive the piece of sugar instead of the whip and move forward. Then some of the stimulus is removed, i.e. the pushing, and the next time the whistle is blown the horse still moves forward, receives his piece of sugar and adjusts the association between the stimuli and the response. And in the end the horse has learned that every time the whistle is blown it is supposed to move forward.

2.3.2 Constructivism

(Greeno & Collins 1996) says that

> understanding is gained by an active process of construction rather than by passive assimilation of information or rote memorization. Conceptual abilities grow out of intellectual activity rather than by absorption of information.

This means that knowledge is gained by thinking about problems and working out a solution for the problem yourself instead of seeing the solution and memorizing it.

Knowledge is constructed in cognitive schemas, and these schemas are build up from already acquired knowledge. According to Jean Piaget (Atheron 2003) the adaption of these schemas are done in two ways. (Borge 2004) has a good explanation.

- Assimilation: fit practice to theory. Complex but familiar external objects are simplified to fit pre-existent categories in your head.
- Accommodation: fit theory to practice. You have to change the ideas in your head to fit the realities of external objects

I will try to give some examples of this kind of schema building. When learning mathematics you first learn to add and subtracts (after learning to count). This can be done by using real world objects and counting them. Four pennies plus two pennies equals six pennies. You construct a schema in your head according to accommodation, and learn how to add small numbers. Then when asked to add larger sets of numbers you use the knowledge already learned and assimilate the knowledge. You break down the complex large number addition into many small number additions. And then you have learned how to add larger numbers.

But you cannot go directly from addition to derivation because you have no pre-existent categories in your head to fit this new complex external object. According to constructivism it is a limited amount of knowledge that can be gained from one cognitive point to another. (Holmboe 2004) says that this knowledge is called the student's "Zone of Proximal Development". Figure 2.1 on the following page tries to show how some knowledge is out of our reach for now, and we have to learn more before being able to learn more complex stuff.

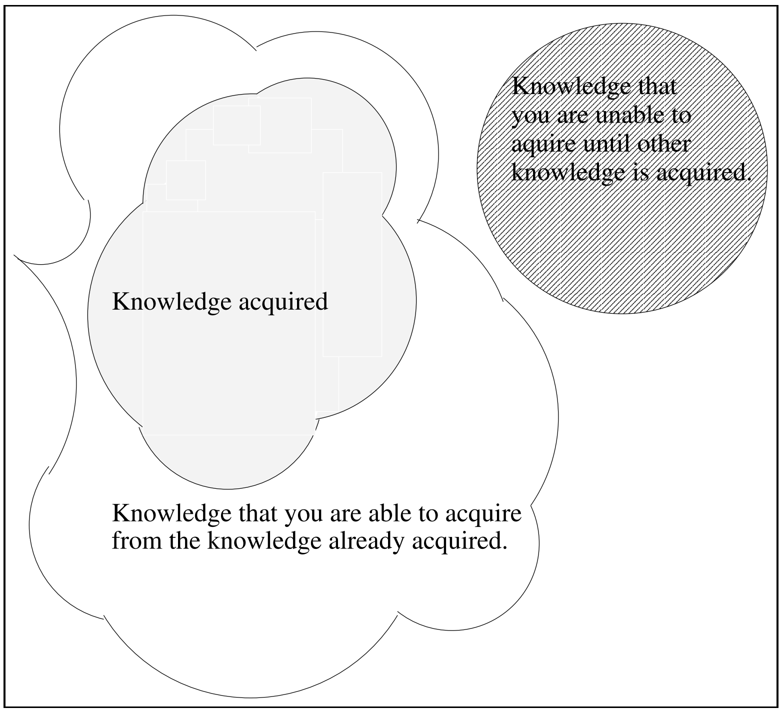


Figure 2.1: What knowledge can be acquired according to constructivism

2.3.3 Situated cognition

Sitting at school in front of a computer working together with friends and colleagues is not the same as sitting on an exam and writing a computer program by hand on a piece of paper. Knowledge is connected with the environment around the learner. In order to learn you have to practice what you want to learn in the same environment you are going to use the knowledge afterward. This way of learning and teaching is the equivalent of a master-apprentice relationship (Greeno & Collins 1996). The apprentice works with the master in the same environment as the master and uses what knowledge he got, and when the apprentice fails to do what he is supposed to do, the master is able to help him through. After a while the apprentice is able to do more and more by himself/herself and needs less and less help from the master. The master is removing the scaffolding (Holmboe 2004) one piece at a time until the apprentice is able to stand on its own feet.

2.4 Why object oriented programming first?

According to (Groven et al. 2003) the dominant approach for teaching object oriented programming is that the students learn the procedural way of programming at first. They learn how to use variables, loops and if-statements. And they learn to use procedures in a procedural way. When using Java, this can be compared to writing everything in the main method. The strange object oriented Java statements like static and class are ignored in the beginning, they just have to be a part of the code in order for it to work.

Then after the basics of programming is taught, object orientation is introduced, and the students are expected to totally change the way they have programmed until now and learn about polymorphism and encapsulation in order to create object oriented programs.

When object oriented programming is taught in this way it is commonly regarded as difficult to understand (Borge & Kaasbøll 2003). The results are that many drop out of the course (Shackelford & Badre 1993). The old way of programming, the procedural way, yielded much better results and was much easier to learn by beginner programmers.

I want to quote (Bergin 2000) as to the difficulty of changing from the procedural programming paradigm to the object oriented one.

> There is nothing especially complex about OOP, any more than there is anything complex about procedural programming. it is just that the world looks completely different in the two paradigms. The experience of the industry is that an experienced procedural programmer will take a year to 18 months to make the switch (Stroustrup 1994). Lattanzi and Henry (Lattanzi & Henry 1996) also report on the difficulty of teaching object-oriented

> principles to students experienced in the procedural paradigm. While the programmers are in this learning mode, they will naturally try to solve problems by decomposing functions and not by discovering objects. Whenever the going gets hard, they will fall back on what they know best: procedural programming. It takes a while for the mind to become re-wired to the new way of thinking. If fact, during this year, the practitioner is likely to build really ugly programs, mixing techniques in an awkward way.

My private experience is exactly that. It takes time to change from one paradigm to the other. But if object oriented programming is the only thing you know about, then this paradigm change never has to happen. And if learning object oriented programming is just as easy or hard as learning procedural programming, why not just start with the object oriented way of thinking at first.

The COOL project, described in section 2.1 on page 12 in this thesis, has a goal of providing an alternative introduction to object oriented programming. According to (Berge & Fjuk 2003), the COOL group believes that:

> It is not object-orientation in principle that cause the problems, but the constellation of artifacts available to learn it.

In this thesis I try to add one more artifact to the constellation: Using Lego Mindstorms to teach object oriented concepts.

Chapter 3

Tools used during the experiment

3.1 Using control-technology as a learning-tool

Control technology enables devices to be programmed to achieve goals. Control technology can be something as simple as using a tape recorder or something as complex as programming a robot to do the dishes.

This kind of technology is often used as a device which enables learning to occur. With this kind of learning, I mean a human learner that uses the control technology to better understand what is going on and not an artificial intelligence that wants to learn by itself.

(Papert 1980) is the inventor of a programming language called LOGO. This language is designed to program floor turtles that can be used to draw different geometrical figures on a piece of paper below them. Young children have been using LOGO in order to learn different aspects of mathematics and acquire some basic programming knowledge. The point of this use of control technology is to enable the learners to see what their programming does and when an error occurs, they will be able to see where in the program the error is according to what the turtle has written on the paper.

In the experiments described in this thesis we use the control technology in the same way as Papert has done. This time with Lego robots instead of floor turtles, but the concept is the same. The subjects are able to see the robot do whatever it is told to do, and if it does something it is not supposed to do, they will know that their program code has an error, and where in the program code the error occurs. And at the same time they will see what the error made the robot do and learn from that experience.

When using this technology to teach object oriented programming, the basic idea is this: When a motor object in the programming code is told to do something, the physical motor object reacts. So when the object in the programming language has received the command forward, the physical motor object will start to go forward. And if it starts going forward at the wrong time, the subject tries to find and fix the error.

Control technology has been used a lot in elementary education. (Denis 1993) is a collection of articles where control technology has been used.

Following is a short description of the different tools used at the experiment at Fjellhamar and Mølladammen.

3.2 Lego and Robolab

Robolab is a program that can be bought from Lego and is used to program Lego Mindstorms robots. It is available both for MacOSX and Microsoft Windows, and is very easily installed onto the computer. The main component of Lego Mindstorms is the RCX.

3.2.1 The programmable Lego brick called the RCX

The RCX is the heart of the Lego Mindstorms robot. It is a big and yellow brick that contains six AA battery elements. On the top of the RCX there is a small LCD display, and in front of the RCX is an infrared receiver and sender. It is also able to play sounds. To get this RCX to communicate with a computer using IR, the computer needs to be connected to an IR-Tower. This tower is part of the Mindstorms for schools package that we used during this experiment.

The RCX has six ports. These are numbered A, B, C and 1, 2, 3. Ports A through C is used when connecting motors and lamps to the RCX, and ports 1 through 3 are used when connecting sensors. See figure 3.1 on the following page for a picture of the RCX. On the front of the picture you see the IR-ports. These must point toward the IR-tower in order to communicate with a computer. You also have to make sure that the RCX is turned on. The buttons on the top are for starting and stopping a program, and for choosing what program to run. The default setting on the RCX is that it can contain up to five different programs.

In the Lego Mindstorms for schools package you also get different motors and sensors that can be connected to the RCX with wires. The sensors include light, temperature, rotation and touch. See figure 3.2 on page 23 for an example. The light-sensor returns a value between 1 and 100 where 100 was the brightest (most light). The touch-sensor returned either the state change from pressed to non-pressed or from non-pressed to pressed. It could also return whether the sensor was pressed or not. The rotation-sensor was a little cross-shaped hole that you could put a cross-shaped shaft in. It would return how many rotations it had done since last reset, or it could return the current angle compared to the last reset. The temperature-sensor returned the current temperature in either Fahrenheit or Centigrade.

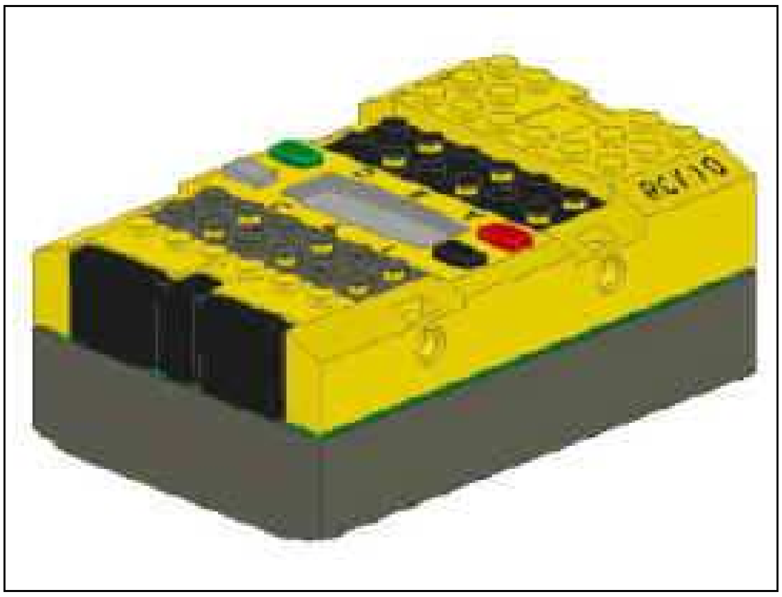


Figure 3.1: The heart of the Lego Mindstorms robot. The RCX

The RCX is pretty heavy and big, so the robot build will have to be pretty sturdy if it is going to hold together.

3.2.2 Robolab Pilot

The Robolab programming environment has two user levels: Pilot and Inventor.

Pilot is divided into four levels, each with increasing complexity and functionality. At the first level you are given a working program consisting of two icons, one to signal the robot to move forward, and one to say how long the robot should move forward. These icons can be exchanged with similar icons, i.e. making the motor go backward instead of forward and changing the time before full stop. See figure 3.3 on page 24 for a picture of a simple Robolab Pilot program. This program tells the motor connected to port A to drive clockwise with the power 3, and the motor connected to port C is told to drive clockwise with the power of 5. They shall do this for 6 seconds, then stop. Start and stop are represented as green and red traffic lights.

On level three and four, the program is divided into more steps, and with each step you say what to do with the three motors and how long until moving to the next step. It can either be for an amount of time, or until something happens with a sensor, i.e. that a touch-sensor is pressed. The graphical interface is shown in figure 3.4 on page 25. You can create more steps with the plus-icon in the top left corner, and browse the different steps

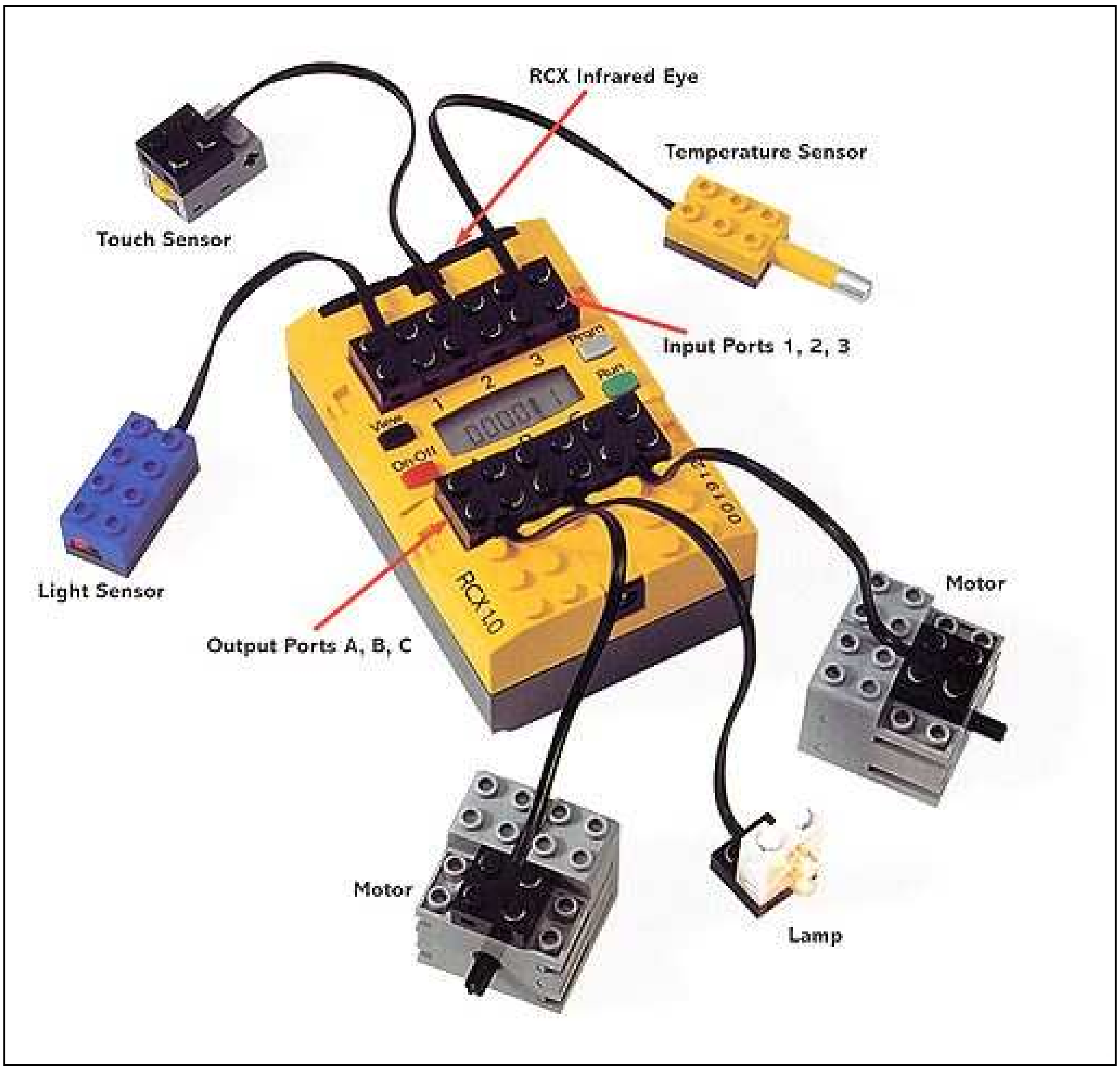


Figure 3.2: An RCX with motors and sensors connected

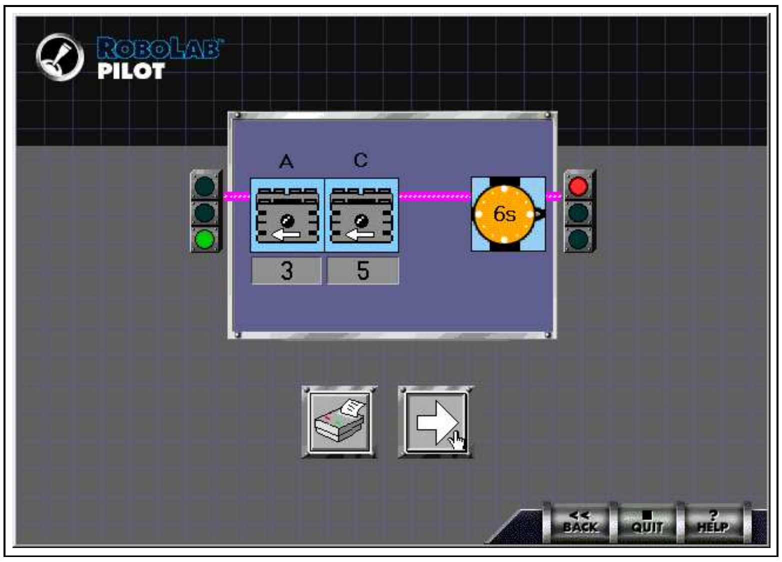


Figure 3.3: A simple Robolab Pilot program

using the red left and right arrows. In order to exchange an icon for another, you just press it once, and your choices pop up. Then just press your new choice. The program can either go through all the steps and then stop, or you can press the purple button to make it go in an infinite loop. When you are satisfied with your program just press the white arrow and the program is transferred to the RCX. There is no way you can make an erroneous program in Robolab Pilot. But you can, of course, still make programs that does something you do not intend.

3.2.3 Robolab Inventor

The Inventor level is more complex. You start with an empty desktop except for the start and the stop icon, represented as traffic lights. To build your program you have a tool case with different icons representing the different actions the robot can perform, and additional icons representing numerical values and such. You drag and drop the icons you need onto the desktop. This can be done in an arbitrary order. The icons can then be moved around on the desktop using the mouse. See figure 3.5 on the following page for a small Robolab Inventor program.

In order to make the first step into the Inventor world less complex, there are four different levels. The lower the level, the fewer icons are available, and the icons available is less flexible and therefore less complex.

To represent the program flow, the user sow the icons together with a thread, from start to finish. To sow the user exchange the normal mouse-arrow with a spool mouse-arrow in the tool box. When the spool is selected

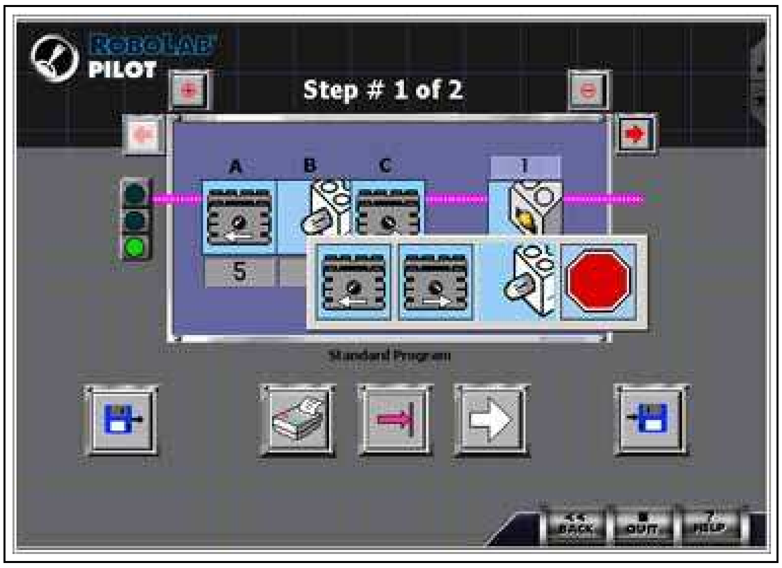


Figure 3.4: A more complex Robolab Pilot program

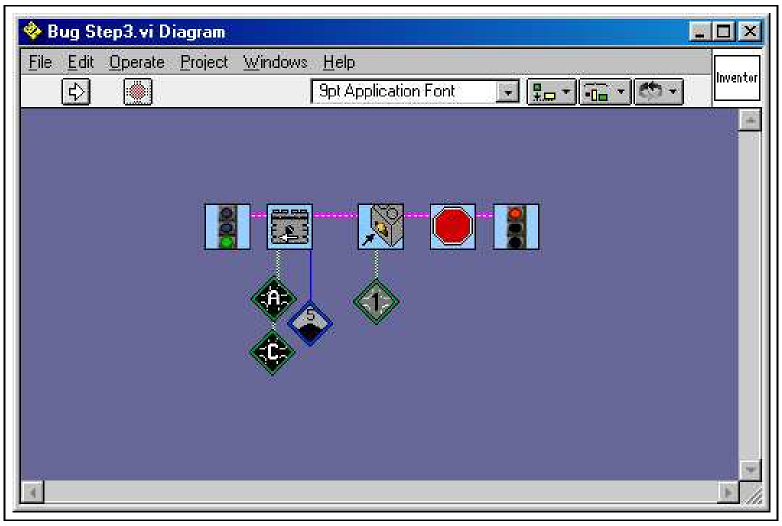


Figure 3.5: A small Robolab Inventor program

and you click an icon on the desktop, you get the start of a line. When you click on another icon, the line is drawn between these icons. I call this way of binding the icons together sowing.

The icon representing the first thing we want the robot to do is sown together with the green traffic light. Then you connect the next icon with the last icon, until you reach the red traffic light. If the connections between the icons are successful, the program is complete and ready for transfer.

If a program is erroneous or incomplete, the picture of the white transfer arrow will be broken. When you press the transfer arrow when its broken, the program will not be transferred to the RCX, instead you get an error message telling you what and where the error is.

Programming functions supported

Figure 3.6 on the next page shows a little program using an if-fork. When run, the robot will turn left as long as the sensor sees something dark, and turn right as long as the sensor sees something light. The program checks if the light-sensor value is higher or lower than 50 (Icon C). If it is higher the topmost route is chosen, and if it is lower or equal the other will be used. Icon B represents the port on which the physical sensor is mounted on the RCX. Icon E represents a motor mounted on port A on the RCX. It will go at a power of 3 (Icon D) and will do so for 1 second (Icon F). The difference between the routes are the direction of the motor. When one way or the other is chosen, the different parts are merged together again (Icon G) and the program continues with only one path.

Inventor also supports jumping from one part of the program to another, using a jump-icon (Icon H) and a landing-icon (Icon A). This is the equivalent of a goto statement. Just going from one place in a program to another. This can be used to make a program run in an infinite loop.

3.3 KarelJ and BlueJ

3.3.1 BlueJ

“BlueJ is an integrated Java environment specifically designed for introductory teaching.”[1] BlueJ is a simplified java-programming tool where the main issue is being able to see a graphical representation of a java-program in a UML-like way. UML is an object-oriented design and analysis language. Its a visual presentation of a computer system where the different parts of a computer program is drawn as squares with a name, a content and the different associations between these squares.

[1]Quote from www.bluej.org

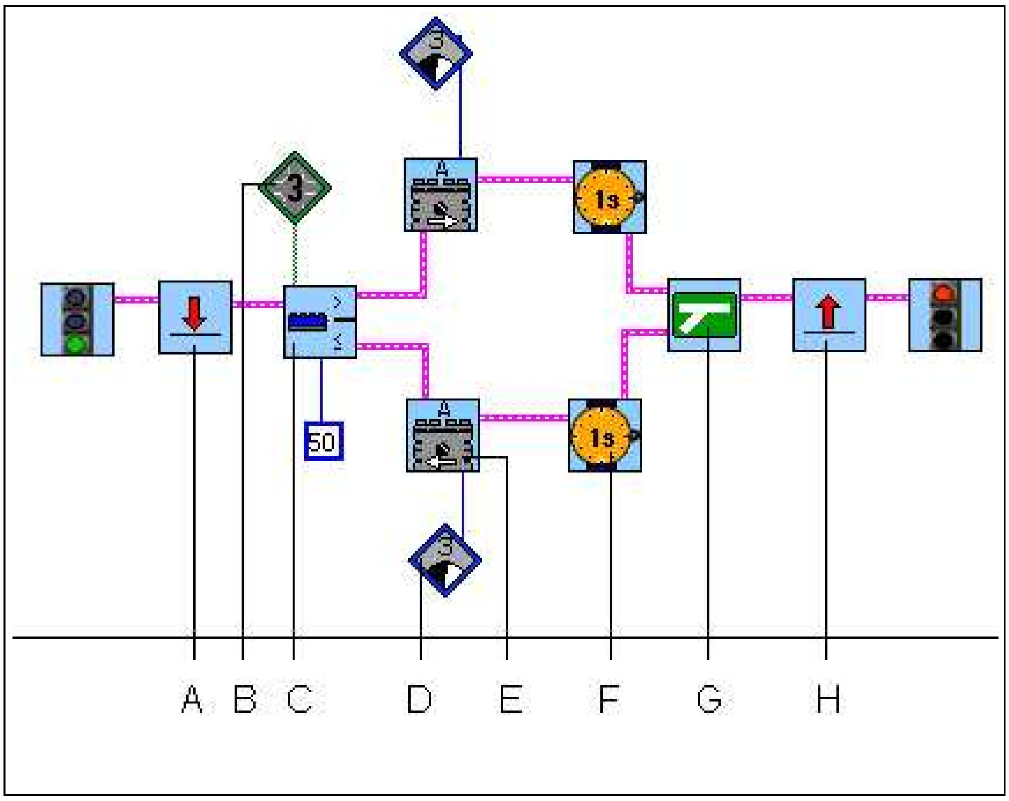


Figure 3.6: A Robolab program using an if-fork and a goto jump.

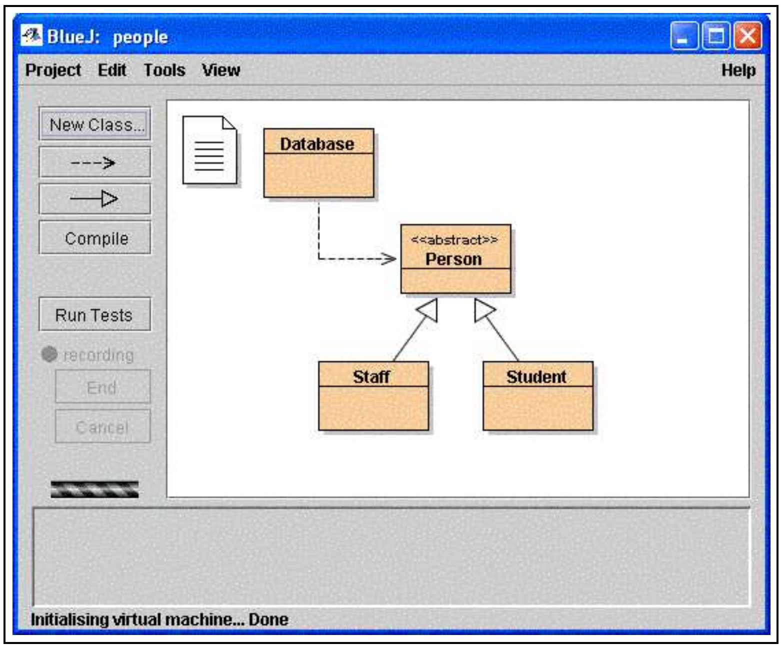


Figure 3.7: A BlueJ example of a person database.

BlueJ is both a Java editor and a tool for visualizing the structure of a Java program. When writing a Java program using BlueJ you create class-boxes that you place on the desktop. When you double-click on such a box, a new window appear with the code for this specific class, just like in any other Java editor. A Java program consists of many classes that are related to each other in some way. This way is represented in BlueJ by arrows. Solid arrows is used to show inheritance and a dotted arrow represents a relationship where one class uses another class. See figure 3.7 for an example of a person database. You can see that Staff is a specialization (a subclass) of a Person and that the Database uses the abstract class Person.

3.3.2 KarelJ

KarelJ is a simulation environment where you program virtual robots on a computer screen. You can use any editor you want to write the Java code, KarelJ is just the engine that shows robots on the computer. A Karel-j program is divided into steps, where a robot does one thing each step. And each step is represented by a line in the java-code.

The KarelJ world is a grid-world with barriers blocking certain paths and some Beepers in the world that the robot may pick up if it got enough room left. The Karel-robot may sense that there is a barrier in front of it, and take a different route to avoid it. It may also sense if there is a beeper underneath it, and pick it up if it wants to.

In order to create flexible robots you need to use normal programming functions like loops and if-statements. Each moving robot on the screen is represented as an object in Karel-j and to make it do something, you had to write a method in the correct robot-class and then invoke this method. You can make several objects of the same class, or several objects of different classes. See (Borge 2004) for a more detailed explanation.

3.3.3 KarelJ and BlueJ together

We wanted to see if a visual representation of the KarelJ program structure would help the programmers understand how the classes and objects are interconnected. We used KarelJ and BlueJ together to achieve this.

BlueJ would show graphical boxes of the classes KarelJ used, and show the relationship between these boxes. When using this combination of BlueJ and KarelJ the programmers are able to double-click on the class they wish to edit, and the program code will be shown in a new window. After the changes are made, the programmer saves and he may open a new class to make changes there too.

When the program is satisfactory, the programmer can right-click on the main class and make an object of this class. And when the programmer wants to run the KarelJ program the programmer tells this object to run. The KarelJ program will be run like any other Java program and the KarelJ grid will appear and the robots will start moving.

3.4 Lejos and Lego Mindstorms

3.4.1 The Lejos API

In the last experiment described in this paper we used a modified version of the Java API for Lego Mindstorms: Lejos. The original lejos (*LEJOS. Java for the RCX* n.d.) API is very complex and contains many features that we do not need for this project. In the original lejos API the program started with a main-method like any other Java program. This was something we wanted to avoid having to teach the children. Another thing we did not feel it was necessary for the children to learn was Java's static term.

The original lejos API is not created as a beginners programming language. Even expert programmers as ourselves had to use a lot of time getting to know the API and how to use it. We used two weeks testing and using the lejos API and used our experience from the tests to create a simpler lejos API that would be more suitable for teaching the object oriented concepts we wanted to teach. We created a template containing a Control class and a Robot class. Using this template the children would be able to create their own robot.

When using the light sensors we found it difficult to find an easy way to program a sensor to listen to changes using a Java actionListener. We decided that all sensors should use active polling instead of using an actionListener. In order to make a robot do something until a sensor value changed, the children had to make a loop and check for a change in value in the loop until the change was found and they could break the loop and continue with the program. This was not an optimal solution but I think this is easier for the tutors to explain and the children to understand.

Following is the template divided into the three parts. Figure 3.8 on the next page is the start of the class file and contains the main-method. This was a necessary evil in order for the children to be able to call their program whatever they wanted. So the only thing they had to do was exchange the <PROGRAM NAME> with the name for their program, and save the file as the same program name.

Figure 3.9 on page 32 shows the control-part of the template. It was in this class the programmers should write the code that told the robot-object what to do. An example of this kind of code would be:

```
volvo.rightMotor.forward(5);

volvo.turn();
```

Figure 3.10 on page 33 shows the robot. This robot should be a representation of the Lego robot the children would build using Lego bricks. They would have to create objects of all the sensors and motors they want to use on their Lego robot, and in this way see how the program they write is executed on the robot.

Using our modified API we should be able to do whatever Robolab is able to do. This modified API consisted of the different objects that could be placed on the Lego robot. In short these objects were:

- Motor
- Touch sensor
- Light sensor
- Rotation sensor
- Display
- Speaker

Assistant

In addition to the objects mentioned above, we had created an assistant object in the control class. The reason for this assistant was that in order to

```
import josx.platform.rcx.*;
import java.util.*;


// Make up your own program name.
// IMPORTANT: Capital letter
// Remember to use "Save as..." and then program name + .java

class <PROGRAM NAME> {
    public static void main(String [] args) {
        new Control();
    }
}
```

Figure 3.8: The beginning of our lejos Template

get the robot to pause between its actions we needed to tell the Java-program to halt a number of microseconds before continuing on the program. This is the same pause that Icon F in figure 3.6 on page 27 represents. This pausing can be accomplished in Java using the class Thread and the method *sleep(milliseconds)*. The program will then halt the given time, continuing the last command while the program waits. This was useful when we wanted the robot to move forward two seconds before doing anything else. The problem with the sleep-command is that it throws an exception which must be caught. And this was not something that we wanted to confuse the children with, so we hid it in an assistant method called wait. We meant that this assistant was not a part of the Lego robot in the same way as the motors, sensors, display and speaker, so we placed it in the control class.

In addition this assistant had a method called *randomNumber(maximum seconds)* that returned a random integer between zero and "maximum seconds". This was useful when programming a more randomized robot. I.e a robot that drove forward and turned, and the length of both the forward driving and the turning was random each time.

3.4.2 JCreator - The text editor)

We wanted an uncomplicated and free Java editor. The only thing we wanted the Java editor to do was to color the Java code so that it would be easier to read and write. Our choice was a small win32 program called JCreator [2].

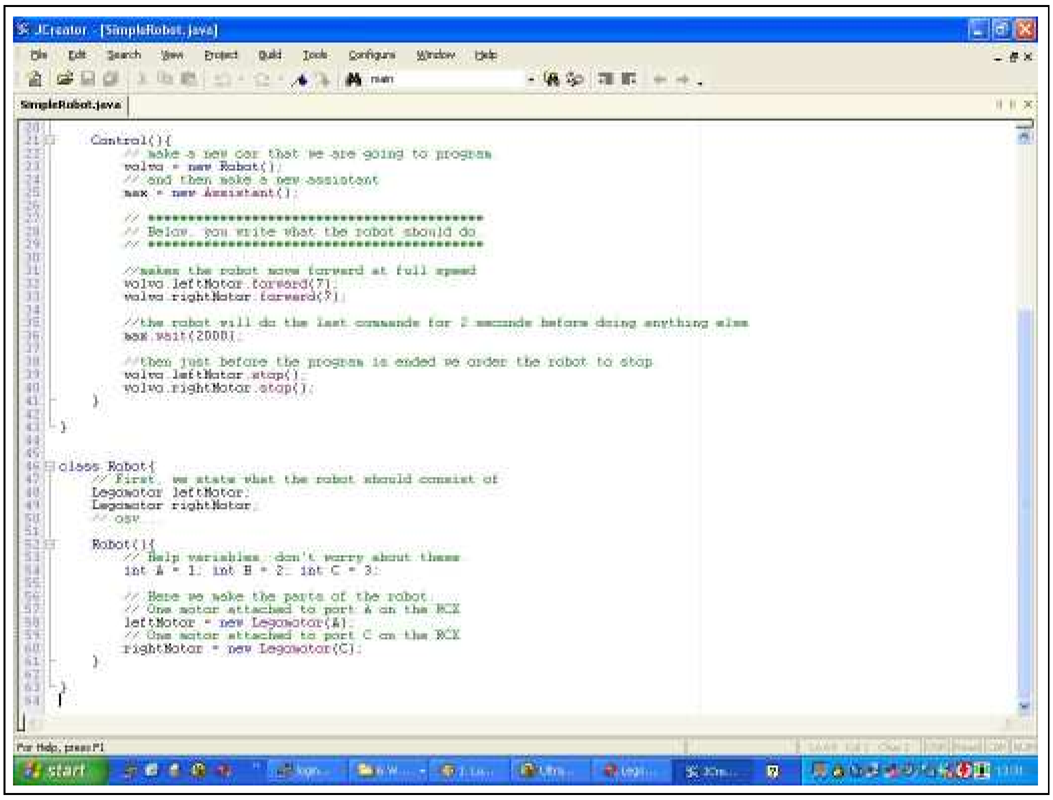
[2]www.jcreator.com

```
class Control {


        // First, we state what the program should consist of
        // In this example, we use a robot called volvo,
        // and an assitant called max
        Robot volvo;
        Assistant max;

        Control(){
                // make a new car that we are going to program.
                volvo = new Robot();
                // and then make a new assistant
                max = new Assistant();

                // *****************************************
                // Below, you write what the robot should do.
                // *****************************************

       }
}
```

Figure 3.9: The Control-part of the lejos template

```
class Robot{

        // First, we state what the robot should consist of
        Legomotor leftMotor;
        Legomotor rightMotor;
        // osv...

        Robot(){
            // Help variables, do not worry about these.
            int A = 1; int B = 2; int C = 3;

            // Here we make the parts of the robot.

            // One motor attached to port A on the RCX

            leftMotor = new Legomotor(A);

            // One motor attached to port C on the RCX

            rightMotor = new Legomotor(C);

            // and so on...
        }
}
```

Figure 3.10: The Robot part of the lejos template



```
    Control(){
        // make a new car that we are going to program
        volvo = new Robot();
        // and then make a new assistant
        max = new Assistant();

        // ******************************************
        // Below, you write what the robot should do
        // ******************************************

        //makes the robot move forward at full speed
        volvo.leftMotor.forward(7);
        volvo.rightMotor.forward(7);

        //the robot will do the last commands for 2 seconds before doing anything else
        max.wait(2000);

        //then just before the program is ended we order the robot to stop
        volvo.leftMotor.stop();
        volvo.rightMotor.stop();
    }

}


class Robot{
    // First, we state what the robot should consist of
    Legomotor leftMotor;
    Legomotor rightMotor;
    // osv...

    Robot(){
        // Help variables, don't worry about these
        int A = 1; int B = 2; int C = 3;

        // Here we make the parts of the robot
        // One motor attached to port A on the RCX
        leftMotor = new Legomotor(A);
        // One motor attached to port C on the RCX
        rightMotor = new Legomotor(C);
    }

}
```

Figure 3.11: A screen-shot of the Jcreator editor

This program is available in two versions. A free version and a professional version. The free one has less functionality than the professional, but the free one still got enough functionality for our needs. See figure 3.11 for a screen-shot of Jcreator.

3.4.3 The compile and transfer tool: RCXDownload

RCXDownload is a free Java program designed for compiling, linking and transferring Lejos programs from a computer to a RCX. It is really just a front end to Lejos, and will not work without Lejos being installed. It has a very easy to use GUI with an "open" button that lets you choose which Lejos program to transfer. After selecting the program, you press a "compile" button that tries to compile the Lejos program. If the program is free of errors the compile is successful and the program is ready for transfer. If it contained errors, the compiler will spot these and display error messages in the RCXDownload GUI. If the program is ready for transfer you can press the "transfer" button, and RCXDownload will automatically link the program and transfer the binary file to the RCX. You see a progress bar in

RCXDownload while the program is being transferred. The RCX plays a sound when the transfer is complete.

It is not necessary to choose which program to compile every time you want to compile a program. Normally you work on a single program for a while, so the program remembers the last file you compiled. The RCXDownload program also has a menu with preferences so you can choose where Java and Lejos are located. See figure 3.12 on the following page for a screen-shot of the RCXDownload GUI.

3.4.4 The Lego help web page

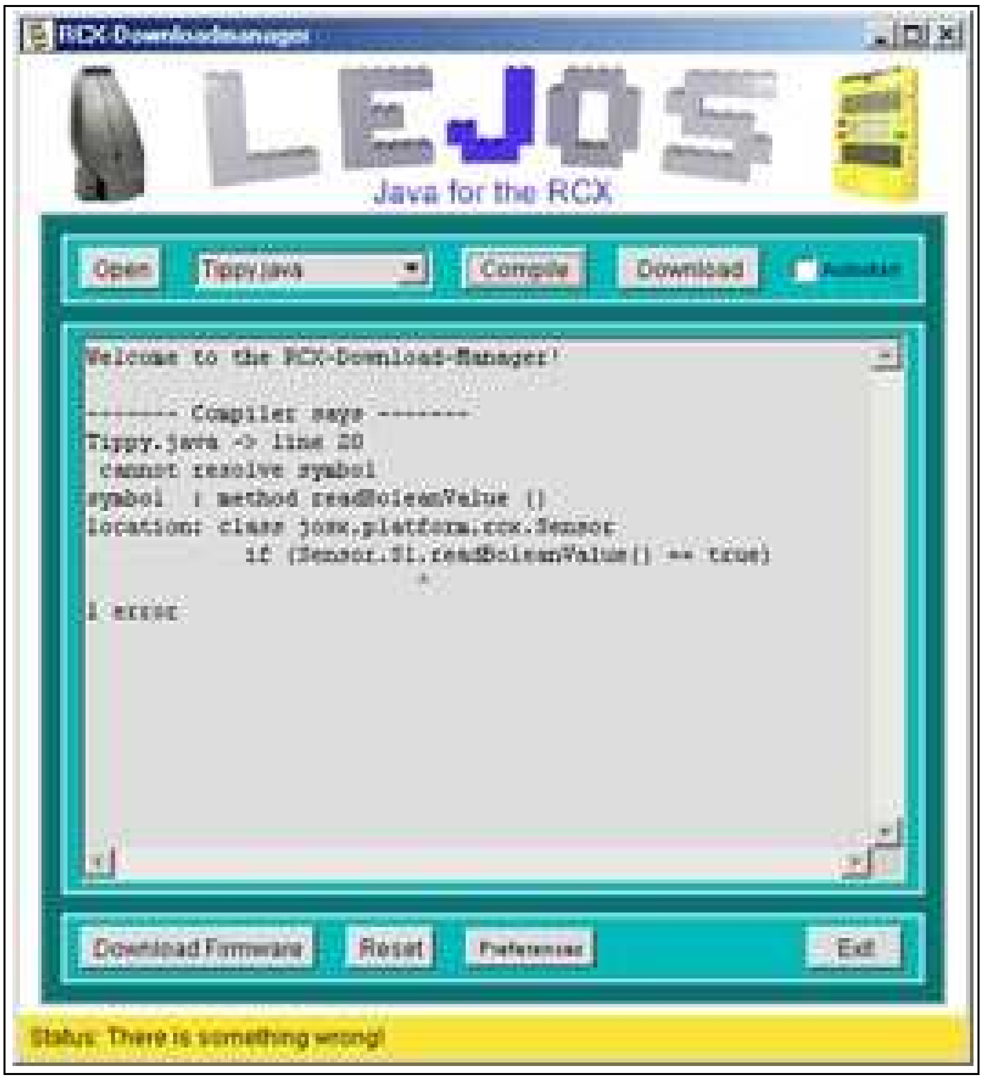
We made a web page containing a lot of information about the experiment. It contained hints for building sturdy Lego robots. Experience from prior Lego building has shown that it is not easy building a sturdy Lego Mindstorms robot. The building instructions was a quite detailed recipe, following the standard Lego instruction booklet way of explaining building. It was divided into different steps, and each step contained a picture of the robot so far, and what Lego parts you needed to finish the step. Figure 3.13 on page 37 is a copy of the start web page. You can see the different menu selections on the left side of the web-page. This web-page is available in the appendix of this thesis.

The web page also contained different programming assignments. Different assignments for using different parts of the Lego robot, for example the motors. These were written assignments with some explanatory pictures. The page also contained some brief explanations of some of the different teaching material. A brief explanation on how to use the editor Jcreator and instructions on how to use the transfer program RCXDownload. The users had access to a complete robot template as well, with a robot with two motors that made the robot drive forward for a little while. When this template was clicked, JBuilder opened this Java file, so the user could save it, compile it and then transfer it to the robot.

The web page had small pieces of code with examples on how to use the different objects that could be placed on the robot. Just click on an object, i.e. Light Sensor, and you could see a small piece of code that did not compile alone, but that could be copy-pasted into a users code. These small code pieces also contained some procedural functions like if-statements and while-loops. They were only briefly explained.

3.5 Tools used when programming a restaurant simulation

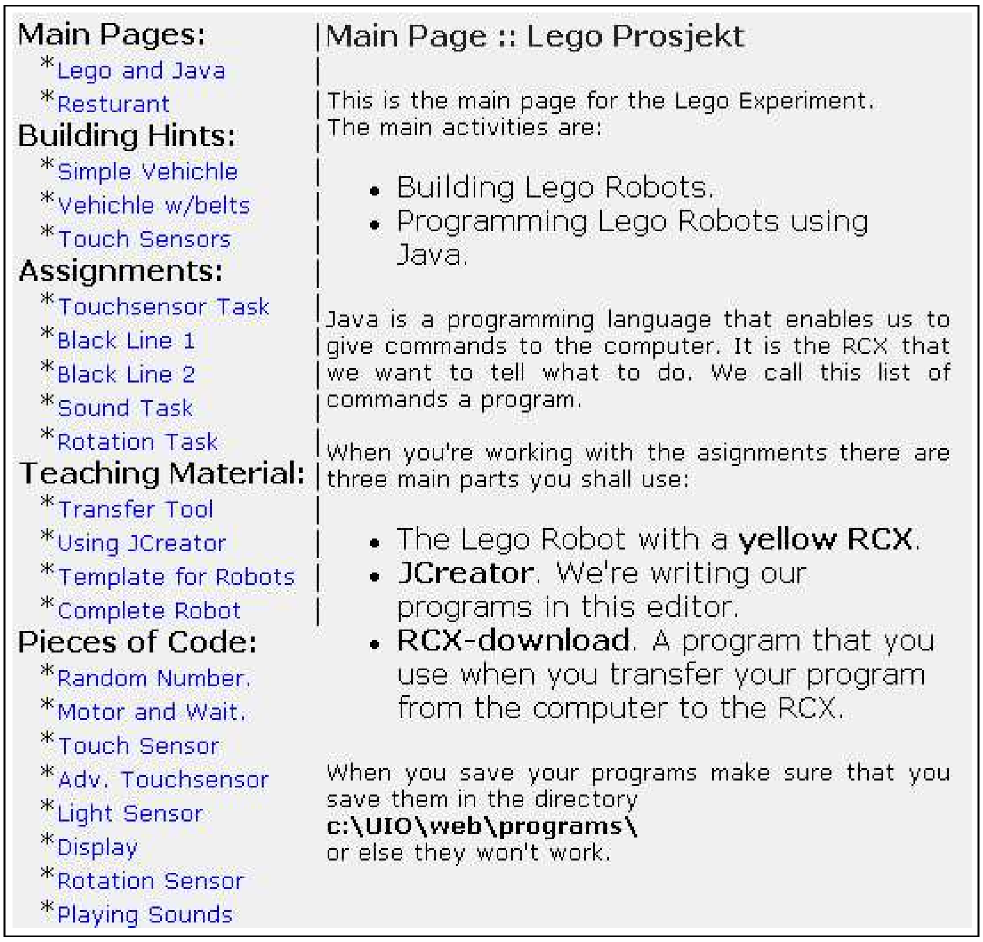
To write “normal” Java code we used JCreator as described above. To compile and run the Java programs made, we used the standard Java tools from



```
Welcome to the RCX-Download-Manager!

------- Compiler says -------
Tippy.java -> line 20
 cannot resolve symbol
symbol  : method readBoleanValue ()
location: class josx.platform.rcx.Sensor
              if (Sensor.S1.readBoleanValue() == true)
                              ^
1 error
```

Status: There is something wrong!

Figure 3.12: A screen-shot of the RCXDownload transfer program



Main Page :: Lego Prosjekt

This is the main page for the Lego Experiment. The main activities are:

- Building Lego Robots.
- Programming Lego Robots using Java.

Java is a programming language that enables us to give commands to the computer. It is the RCX that we want to tell what to do. We call this list of commands a program.

When you're working with the asignments there are three main parts you shall use:

- The Lego Robot with a **yellow RCX**.
- **JCreator**. We're writing our programs in this editor.
- **RCX-download**. A program that you use when you transfer your program from the computer to the RCX.

When you save your programs make sure that you save them in the directory **c:\UIO\web\programs\** or else they won't work.

Figure 3.13: The front page of the Lejos help web page

Sun Microsystems. We only needed the Java compiler and the Java runtime environment. These are just two command-line applications that comes with the standard Java package.

javac —– the compiler
java —– the virtual machine that runs Java-programs.

To make these tools as easy as possible to use, we made two small scripts, one for compiling the java-code and display the error-messages, and one for running the Java programs and displaying the results. The users only had to double click on the script they wanted to run, and look at the results.

The Restaurant API consisted of some custom made Java classes used in a Restaurant simulation. These classes and their methods were:

- Person(Name of the person, sex of the person)
  When a person is created he needs to have a name and a sex
- Guest extends Person
  - chooseFromMenu(mainCourseNumber, dessertNumber)
    Makes the guest choose a main course and a dessert
  - mainCourse()
    returns the number of this persons main course
  - dessert()
    returns the number of this persons dessert
- Chef extends Person
  - cookFood()
    Cooks all the food ordered
- Waiter extends Person
  - changeNumberOfSeats(table, number of seats)
    lets a waiter change the number of seats at a table. The table will be reset
  - placeByTable(guest, table)
    places a guest by the given table
  - receiveOrder(guest)
    gets the order from a guest
  - getMainCourse()
    gets a random main course from the kitchen
  - getDessert
    gets a random desert from the kitchen

  - findCorrectGuest(table)
    finds the guest at the table that ordered the course just gotten. This method returns the person that ordered the course
  - deliverCourse(guest)
    Delivers the course just gotten to the guest

- Table(numberOfSeats)
  The table is placed in a restaurant and has a number of seats
- Menu()
  The menu is created in a restaurant and contains the different courses and menus

There are two additional classes should be modified in order to create a correct restaurant simulation. These were:

- Control()
  This class only existed in order to make a Restaurant object and for the users to write their restaurant simulation code. It had the same function as the control-class in the Lego Robot programming. It might have been more natural to write the execution code in the restaurant class, but since we had written the execution code in the control class when programming Lego robots, we wanted to do the same in the restaurant assignment.
- Restaurant()
  This was the restaurant that was simulated. Here the tables, the waiters, the chef and the menu was placed. It resembles the Lego robot class.

  This is an API that is handed out to the users at the start of the assignment.

Chapter 4

Method

4.1 Method

Following is a short description of what methods were used during the experiments at Fjellhamar and Mølladammen. Chapter 4.4 on page 44 and chapter 4.5 on page 46 gives a detailed day by day description of what was done during the experiments. This thesis will focus on the results from the experiment at Mølladammen where Java was used to program Lego Mindstorms robots. The reason is that Fjellhamar was more of a pilot experiment, and the results from this pilot project is less relevant for this thesis than the results from the experiment at Mølladammen.

4.1.1 The Subjects

The subjects were 27 children at Fjellhamar and 28 children at Mølladammen. This was one school class at each school, and the participating classes were picked by random. The children at Fjellhamar were 11 years old and the ones at Mølladammen were 14. As explained at the start of the thesis, the 11 year old children were really 11 to 12 and the 14 year old children were really 14 to 15 years old. Very few of the children had any prior experience in programming, and those few that said that they had some experience had never used Java as a programming language.

4.1.2 Experiment start

At Mølladammen the experiment started with a short introduction where they were taught some object oriented concepts and guidance to use the Java API called Lejos. We wanted to throw the subject into complex problem solving, without a long pre-lecture. At Fjellhamar we started with a game so that the children would get to know their tutors.

4.1.3 School relevance

The experiments were placed in the natural science part of the schools curriculum. So a while the experiment was performed, some groups were taken from their seats and given a solar panel and a motor. And their task was to make their robot go without using the RCX. This was most use as an excuse in order to get to use the school classes during these experiments.

Another reason for introducing this kind of technology to these children was that computer science is going to be part of the base curriculum at elementary schools. And according to the norwegian politicians this is going to happen in the near future.

4.1.4 Data collection

At first the study's purpose was to survey and explore the way people learn and practice object oriented programming with a custom made Java API called Lejos. This was done by first going through a pilot study at Fjellhamar where we tested the Lego equipment and the equipment for recording the data from the sessions. This was done by cameras that was able to film both a group of children and their computer screen at the same time. The discussions were also recorded with a microphone. Then at Mølladammen we would use what we learned from the pilot experiment and find out how much object oriented programming we were able to teach the children in our short time.

When we later would look at the data recorded, we would be able to see what the subjects were doing in their programing and simultaneously see the subjects' discussions. This would be very helpful when trying to determine if the subjects understood what they were doing. After the study there was approximately three hours of video from the Lego Robot programming with Java at Mølladammen 4.5 on page 46, and two hours of video from the interviews the last day at Mølladammen 4.5.3 on page 48. From Fjellhamar there was about ten DV-cassettes of one hour video each. In addition there was a lot of hours recorded with the streaming server as well, but I never took the time to look that them.

4.1.5 Use of Control Technology

We used Control Technology to help the subjects see the connection between the real world and the computer program they wrote. We wanted them to see with their own eyes, what effect the program they wrote, had on the object they were programming, in this case, a Lego robot.

The last day both at Fjellhamar and Mølladammen we removed the Lego. We wanted the children to use what they had learned using the Lego and program something similar but different. At Fjellhamar we kept the robot domain, but instead of programming real life Lego robots the children had to program virtual robots on the computer screen. At Mølladammen we changed even more. We removed both the Lego and the idea of robots. At Mølladammen the last day was used to program a restaurant simulation.

4.1.6 Research method

We used qualitative research methods. Qualitative research methods were originally developed in the social sciences to enable researchers to study social and cultural phenomena (Borge 2004). Different methodologies are used in different parts of science. I believe that the study lays closer to the soft aspect of science, also the closest to social science.

Qualitative data sources include observation and participant observation, interviews and questionnaires, documents and texts, and the researcher's impressions and reactions(Borge 2004). The observational studies consisted of recording video, both covering the subjects when they programmed alone and with assistance. The recorded material consisted of sound and computer screen of the students' computers, while they programmed.

In order to being able to use a more quantitative research method we would have needed a more specialized assignment. We could have measured how long time the different groups had taken in order to complete the different assignments, or how many groups got a correct answer on the first try. During the experiments we did not measure any of these differences between the groups, so I do not have any quantitative data that I can use.

4.1.7 Short interview

In the closing stages of the experiment at Mølladammen, four groups of three subjects were handpicked and given small assignments related to object orientation. Two of the groups were picked because they showed a lot of understanding during the Lego robot programming. The other two groups were picked at random. I talked to the groups one group at a time for about half an hour. The idea of this interview was to map the subjects deeper understanding of object-oriented thinking. They were given papers on which they were going to sketch the different objects and inscribe sentences with the terminology they were introduced to earlier. This last part of the experiment was not something we had planned and used a lot of time preparing. I just picked a group while they were programming the restaurant and asked them to join me in a more private room with their Lego robot and their laptop. Then I asked them questions that I felt would give me an idea of how much they had learned about programming and object oriented concepts. All of the interview sessions were recorded using a video camera.

4.1.8 Point of study

The studies evaluate the subjects' behavior and quality of product, but is not intended to result as teaching guide in programming. However we are doing experimental studies with different subjects and new learning styles. We hope the answer would aid us in understanding of teaching object oriented concepts and show us that young adults also enjoy programming in object oriented languages, and programming in general.

4.2 Pedagogical Invention

We wanted to use Lego Mindstorms as the mean for teaching object-orientation. With the use of a custom made Java API called Lejos we wanted to use Control Technology(Valcke 1993) to see if a physical representation of an object helps in understanding how the Java representation works. If the subjects in their Java program writes
robot.rightMotor.forward(),
they will be able to see both the physical robot, and the attached physical motor. When the physical representation of robot.rightMotor starts to go forward the subjects will see the direct correspondence between the robot.rightMotor object and the physical motor on the right on the physical robot.

Lego has been used as Control Technology in a lot of other subjects, such as physics and math. Lego Mindstorms is really just an advanced version of the turtles in Logo(Papert 1980). Logo has been used to teach geometry and other math-aspects especially to children and young adults. It has also been used to some programming aspects i.e. recursion(Papert 1980). According to our knowledge this is the first time that Lego has been used to teach object-oriented concepts.

4.3 Experiment Overview

During this experiment we used four different tools. The different tools are:

- Programming Lego Mindstormsrobots with Robolab
  - This was used at Fjellhamar with 11 year old children
  - This was used for three days
- Programming virtual robots on the computer screen with KarelJ and BlueJ
  - This was used at Fjellhamar with 11 year old children
  - This was used for one day, the last day

- Programming Lego Mindstorms robots with JCreator and Java
  - This was used at Mølladammen with 14 year old children
  - This was used for two days
- Programming a restaurant-simulation with JCreator and Java
  - This was used at Mølladammen with 14 year old children
  - This was used for one day, the last day

4.4 Day by day at Fjellhamar

Following is a description of what was done at Fjellhamar and Mølladammen. This is to give the reader an overview, and to more easily be able to understand the results from the experiments. This sub-chapter is about Fjellhamar and the next is about Mølladammen.

4.4.1 The Experiment

This experiment was meant to be a pre-experiment prior to Mølladammen. We wanted to see how the Lego worked, and we wanted to test the other technical equipment as well, like the cameras and streaming computer. The goal in this experiment was not to teach the children any object-oriented concepts. We did not think that this would be possible with the Robolab tools.

The experiment took place over four full school-days where we had three days after each other the first week, and then returned for one day two weeks later. Their class-teacher had divided them up into 9 groups with 3 children in each group. We would have had more groups if we had more Lego boxes. There were no mixed groups, only groups with boys only or girls only. This was probably a good thing, since they were 11 years old.

Each day started with a physical game to build up the teamwork between the children. One very cool game was to see how few chairs the whole class could stand on at the same time. The game started with lots of chairs, and then a one chair was removed at a time, and the children had to help each other stand on the chairs.

A couple of times a day we would have a gathering in a room and sit in a circle. The children were then asked different questions about that they were doing and how and what they were thinking of what they were doing. They had to explain the robots they had built and talk about the different parts and their properties. We wanted them to answer what made a vehicle a vehicle, and what parts of a vehicle was common and what parts were unusual

4.4.2 The first day

The first day we wanted the children to play with Lego, build their own Lego-house and then explain what the house consisted of, and the properties of these parts. This was the start-experiment and did not include either Robolab, motors or sensors.

The next task was building a Lego-robot that included the RCX, and used wheels. Then they could program these vehicles using Robolab. Their first programming assignment was making the robot they built do something. The children's second assignment was to make the robot go forward, then turn, then go backward. The trick was to understand how to turn the vehicle. Either making the motors go opposite ways, or stopping one motor and letting the other continue, or setting the different motors at different powers, so that one would go faster that the other.

We had started their computers with the Robolab-program so the first thing they saw was 2 icons and a "transfer" button. All they had to do to transfer this program to the robot was pressing this button and making sure the robot was switched on. This was the first level of the Pilot version of Robolab. Then the groups advanced through the different levels of Robolab Pilot. At the higher versions the program lets you use sensors like the touch and light sensors.

4.4.3 The second and third day

The second day we introduced Robolab Inventor. Inventor is a lot more flexible than Pilot, and some of the programming tasks required this flexibility. We had programmed a demonstration robot to follow a black line on the floor. In order to make that kind of program, you need to use an if-statement, and the Pilot version of Robolab does not support that. The demonstration robot showed the children the possibilities they had when using Lego Mindstorms.

In the middle of the second day the children were told that the rest of the time should be used solving a bigger project that they themselves should decide what consisted of. The groups got pieces of paper to write a description of their project robot on. They were also supposed to draw a picture of the project robot. And after this "plan" had been approved by us, they were allowed to spend the rest of the time building their robot from the Lego bricks, and program it to do what they described in their project plan.

The possibilities of their projects were only restricted by the boundaries of the RCX. It only had three ports for motors and three ports for sensors. Therefore they could not build a robot with two motors for driving and turning, and two arms operating separately.

They had to spend some time getting to know how the light sensor and the touch sensor worked. They had seen the light sensor in use on the demonstration robot, but they did not know exactly how it worked.

As the groups worked with their projects, the teachers helped the groups that asked for help and tried to see how they were following their plan when building their robot.

4.4.4 KarelJ and BlueJ the last day

When we returned two weeks later we did not bring any Lego. This time we wanted the subjects to program robots on the computer screen using KarelJ and BlueJ. The goal was to keep the programming domain and see if they could use anything they had learned programming the physical Lego robots. They were divided into the same groups as when programming with Robolab.

Their programming tasks were to make a subclass of a simple KarelJ robot and create new methods in the new and more advanced robot. In BlueJ this is shown graphically in a UML like way (see figure 3.7 on page 28). They would program the robot to navigate some pre-made KarelJ worlds, and picking up the beepers in the maze.

At first we only wanted the children to write the program in a procedural way. Just instructing the robot line by line what to do. Then we introduced more flexible ways of doing things, with while-loops and if-statements. They would then be able to create flexible robots that felt their way through the maze and managed to pick all the beepers.

We even tried programming a messing-robot that put out beepers randomly in the world, and they had to make a flexible cleaning-robot that had to clean up the mess. This is a very challenging task, and our messing robot was not very good and sometimes it would create a mess that was impossible to clean up.

4.5 Day by day at Mølladammen

This school is a junior high school where the children were 14 - 15 years old. It was a well funded school. We were allowed to borrow 10 brand new laptops for the project. The study's purpose was to survey and explore the way people learn and practice object oriented programming. We had two cameras and the possibility to film both a group and the groups computer screen at the same time. This was the same equipment we used at Fjellhamar.

4.5.1 The Experiment

All the sessions were filmed. We had a camera that filmed one of four groups, that we thought would be the most interesting groups. The streaming computer also had a direct feed of the computer screen of one of the groups, and a switch-box was used to select one of the four screens for streaming. So when we look at the material later on and see both one of the four groups and their computer screen at the same time.

4.5.2 The first two days

We started the first day just playing with the Lego and building a robot that included the RCX. Then the children started using the web-page we had created. It hopefully contained everything that they needed to know. This web-page also contained step by step guides in how to create a sturdy robot that used belts instead of wheels. In order to get started we had made a simple start-robot code that just made the robot drive forward for two seconds. Then the children, using an printout of our API had to program the robot to do whatever they wanted.

There were nine groups of children with three people on each group. And one group had one person extra. We had learned that it would be pointless to try to get them all on tape, so we concentrated on four groups. Most of the results are from the videos of these four groups. They were also the groups that got the most attention when asking for help.

In order to use i.e. a light sensor, they had to make a new light sensor object in the Robot class, and then call this light-sensor object from the control class. Or they could make a method in the Robot that used the "local" sensor. We wanted see observe the way they used the dot-notation to get to the right methods in the right objects.

The children had a lot of assignments to solve, with increasing difficulty. The first assignments only required the children to copy and paste from code already given, but later on when the assignments got more complex, they had to learn how to use the different sensors, and how to write something to the LCD display. One of our goals was that they would create methods in the robot class that they could call from the control class. An example of a method of this kind would be a turn method, that made the correct commands to make the robot turn 180 degrees.

One of the assignments the children had to do was to create a robot that followed a black line. A fully functional robot that we had programmed was used as a demonstration. To program this robot you had to use both a while loop and if statements. In order to make the assignment a little easier we created a similar assignment that the children could solve first. Create a robot than can find its way out of a black circle with a little opening. They were supposed to create a robot that drove forward until a light-sensor saw the black tape, then back up a little, turn a little (maybe even turn a random number of seconds) and then continue forward again. The good thing about this exercise as a start is that it does not require an if-test with different outcome-paths. Just making the robot go forward until it sees the tape is easier than having two sensors that makes the robot do different things.

4.5.3 The third and last day: The object oriented restaurant

The last day we removed the Lego. We wanted the children to do something completely different. We wanted them to program a restaurant simulation. They would still use Java in approximately the same manner, but there would be no live action in which the results would be shown. All they would get is some text on a computer screen.

Following is the assignments that were given to the children along with a paper copy of the restaurant API:

The task in this assignment was as follows:

Four guests arrive at the restaurant. You have to come up with names yourselves. The guests want to be seated at a table and eat a nice meal with dessert. You will have to decide what they will eat from the menu.

Problem 1

When they arrive there is only one table free. There are only two chairs at this table, but they are a party of four. When they have been seated they will choose what to eat from the menu, the waiter will take the order and the chef will cook the food.

Problem 2

The food is cooked simultaneously but will be done in random order. The waiter have to find out who ordered what in order to delver the correct course to the correct guest. Remember that the guests have to eat their main course before they can eat dessert.

We used the same groups as we had done the previous days. The children were told to look at the picture of the restaurant[1]. See figure 4.1 on the next page. They could see this picture when they clicked one of the links in the help web-page. Before they started programming the restaurant their assignment was to find a name for this restaurant, and write down which objects this restaurant consisted of.

Then the different objects of the restaurant was written down and these were discussed in the class. Now the children were ready to start programming their own object oriented restaurant using Java.

[1]This picture of a restaurant is the same Nygaard used when he argued for the objects first way of teaching object oriented programming. He meant that in order to teach object oriented programming, you had to start out with a sufficient complex example, so that the programmer had to use object oriented structuring in order to solve the task ahead.

OBJECT-ORIENTED REASONING AND WORKING STYLE
Restaurant
UKRUT–Pål Hansen
Seleksjonsfragmentert perspektivtegning
m.h.p. framework design
i objektorientert environment.
for Kristen Nygaards foredrag hos Genera

Figure 4.1: A picture of Nygaards object oriented restaurant

The assignment was to get the guests to order the food they wanted, giving the order to the waiter who would take the order to the chef. The chef would then cook the order and the waiter would take the different courses back to the people that ordered them. The challenge was that the courses were finished at random, and the waiter had to find out who ordered the different courses using his own method: *findCorrectGuest(table)*.

When the program was compiled successfully and run, it printed out the simulation step by step. Figure 4.5.3 on page 53 shows an example of such a printout. Figure 4.5.3 on page 53 shows an example where the guests has received the wrong courses.

The Restaurant assignment was meant to be difficult and we had not counted on every group finishing the assignment. The children only had one day on this assignment. To make this assignment as similar to the Lego robot programming assignment, we used a control class to write the actual code in this assignment as well.

Figure 4.2 on the following page shows the typical start of a control class in the restaurant assignment. Figure 4.3 on page 52 shows a typical restaurant class in the same assignment.

4.5.4 Four interviews the last day

During the restaurant assignment the last day I wanted to interview four groups about what they had learned during the experiment. I chose two groups that I knew had performed pretty well during the Lego robot programming assignment, and two random groups. They were taken to another room where I had placed some chairs, a little table and a video-recorder. They brought their Lego robot and the laptop they had used along with them. These interviews took 30 minutes and after they were done they went back to the restaurant assignment.

The interviews started with some smalltalk about how they felt the experiment had been. Then I gave them a piece of paper and asked them to draw their Lego robot using boxes and lines. I started them off by drawing a square that I called control and a square that I called robot. Then I drew a line between them, connecting them together. Then I wanted them to draw the different robot parts on this drawing in the same manner. Figure 4.6 on page 55 shows a drawing made by the children during this interview.

At the same time they were drawing I asked them different questions to try to find out how much they had learned during the experiment, and if this knowledge could be transferred to this assignment, which was very different from the one they had just participated in. A lot of the questions was about how they would write different call sentences in order to invoke different methods. I wanted to see if they were able to generalize the dot-notation they had used when programming the Lego robot in Java.

After I was satisfied with their drawing of the robot I wanted them to

```
class Control {

  Restaurant objecta;
  Person per;
  Person lise;
  Person kjell;
  Person hanne;

    Control() {

      objecta = new Restaurant();
      per = new Guest("Per", "male");
      lise = new Guest ("Lise", "female");
      kjell = new Guest("Kjell", "male");
      hanne = new Guest("Hanne", "female");

      objecta.butler.changeNumberOfSeats(objecta.table4,4);
      objecta.butler.placeByTable(per, objecta.table4);
      objecta.butler.placeByTable(lise, objecta.table4);
      objecta.butler.placeByTable(kjell, objecta.table4);
      objecta.butler.placeByTable(hanne, objecta.table4);
      ...
      ...
```

Figure 4.2: The start of the Restaurant control class

```
class Restaurant {

  Waiter butler;
  Chef chef;
  Menu menu;
  Table table1;
  Table table2;
  Table table3;
  Table table4;

  Restaurant() {
    butler = new Waiter("Eva", "female");
    waiter = new Waiter("bob", "male");
    chef = new Chef("Tor", "male");
    menu = new Menu();
    table1 = new Table(4);
    table2 = new Table(2);
    table3 = new Table(5);
    table4 = new Table(2);
  }
}
```

Figure 4.3: The typical Restaurant class

```
Per is added as a guest
Lise is added as a guest
Kjell is added as a guest
Hanne is added as a guest
Beef is ordered
Cake is ordered
Salad is ordered
Jelly is ordered
Salmon is ordered
Chocolate pudding is ordered
Salad is ordered
Ice cream is ordered
The chef cooks the food
Per has received Beef as a main course. And this is correct.
Lise has received Salad as a main course. And this is correct.
Kjell has received Salmon as a main course. And this is correct.
Hanne has received Salad as a main course. And this is correct.
Per has received Cake as a desert. And this is correct.
Lise has received Jelly as a desert. And this is correct.
Kjell has received Chocolate pudding as a desert. And this is correct.
Hanne has received Ice cream as a desert. And this is correct.
```

Figure 4.4: An example of a correct result from the restaurant program

```
Per has received Salad as a main course. Incorrect. Per ordered Beef.
Lise has received Salmon as a main course. Incorrect. Lise ordered Salad.
Kjell has received Salad as a main course. Incorrect. Kjell ordered Salmon.
Hanne has received Beef as a main course. Incorrect. Hanne ordered Salad.
Per has received Cake as a desert. And this is correct.
Lise has received Chocolate as a desert. Incorrect. Lise ordered Jelly.
Kjell has received Jelly as a desert. Incorrect. Kjell ordered
Chocolate.
Hanne has received Ice cream as a desert. And this is correct.
```

Figure 4.5: An example where the guests received wrong courses

draw an object oriented house on another piece of paper. I gave them this task because I wanted to see if they had gotten a general understanding of how the different parts of their program was connected. I also wanted to see if the knowledge gained from programming the Lego robots could be transferred into a more general understanding. Figure 4.7 on page 56 shows a drawing made by one group of children. The results from these interviews are found in chapter 5 on page 57 along with the other results from the experiments.

4.6 Going through the collected data

When the experiments were complete I had about half a years time until the masters thesis was due. I had a feeling that I would not have the time to go through all the recorded material both from Fjellhamar and Mølladammen. Since this thesis was going to be about teaching object oriented concepts using Lego Mindstorms, I chose to focus on the material from Mølladammen.

I quickly browsed through some of the material from the group sessions at Fjellhamar and transcribed this, and some of that material is used in this thesis. But all in all there are very few results from the experiment at Fjellhamar described in this thesis.

From the experiment at Mølladammen I had the video recordings and in addition I had some notes taken during the experiments and the code the groups had written, both for the Lego robot assignment and the restaurant assignment. I also had the video tapes from the last day interviews with four of the groups.

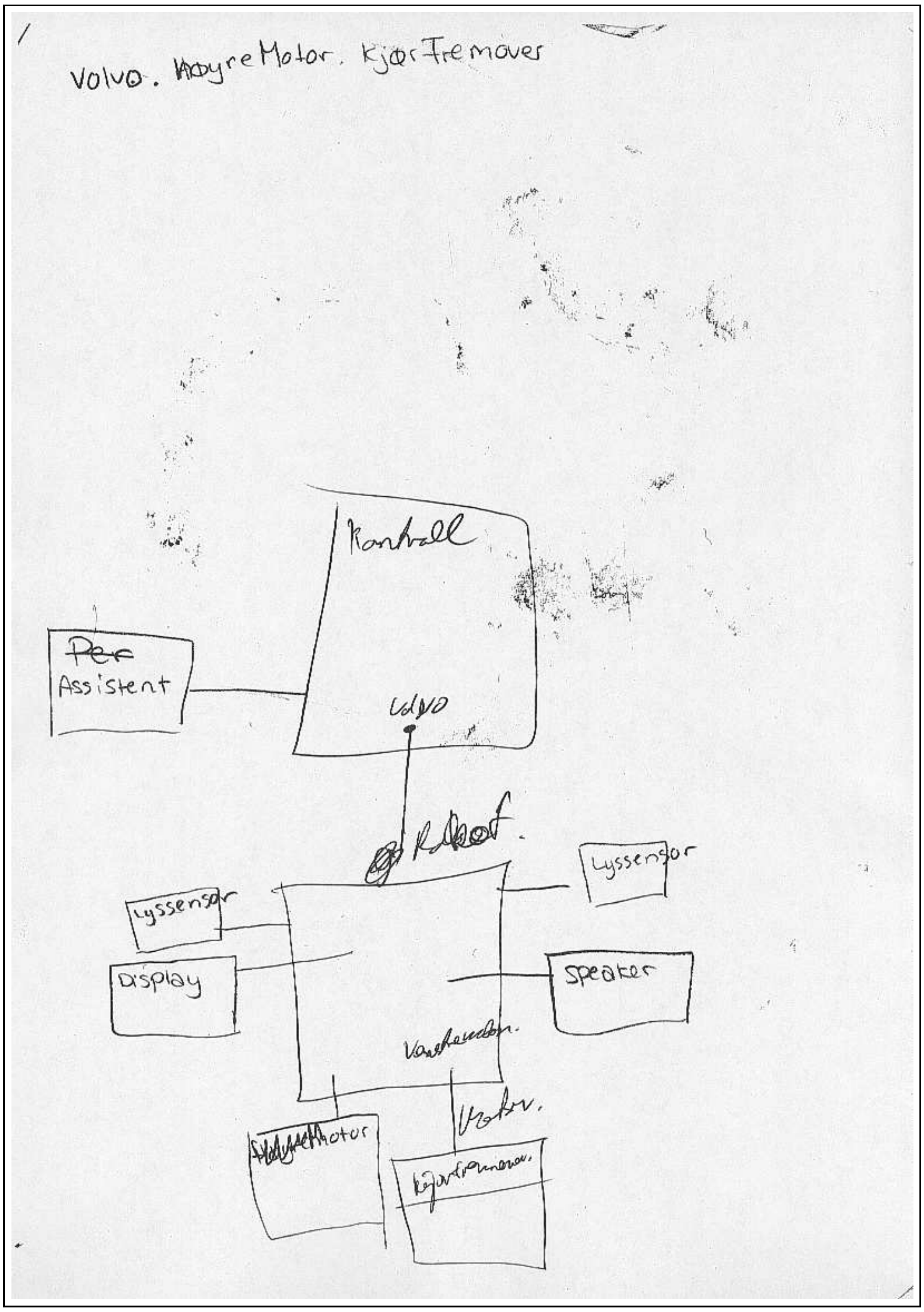


Figure 4.6: A drawing of an object oriented robot program, drawn by the children

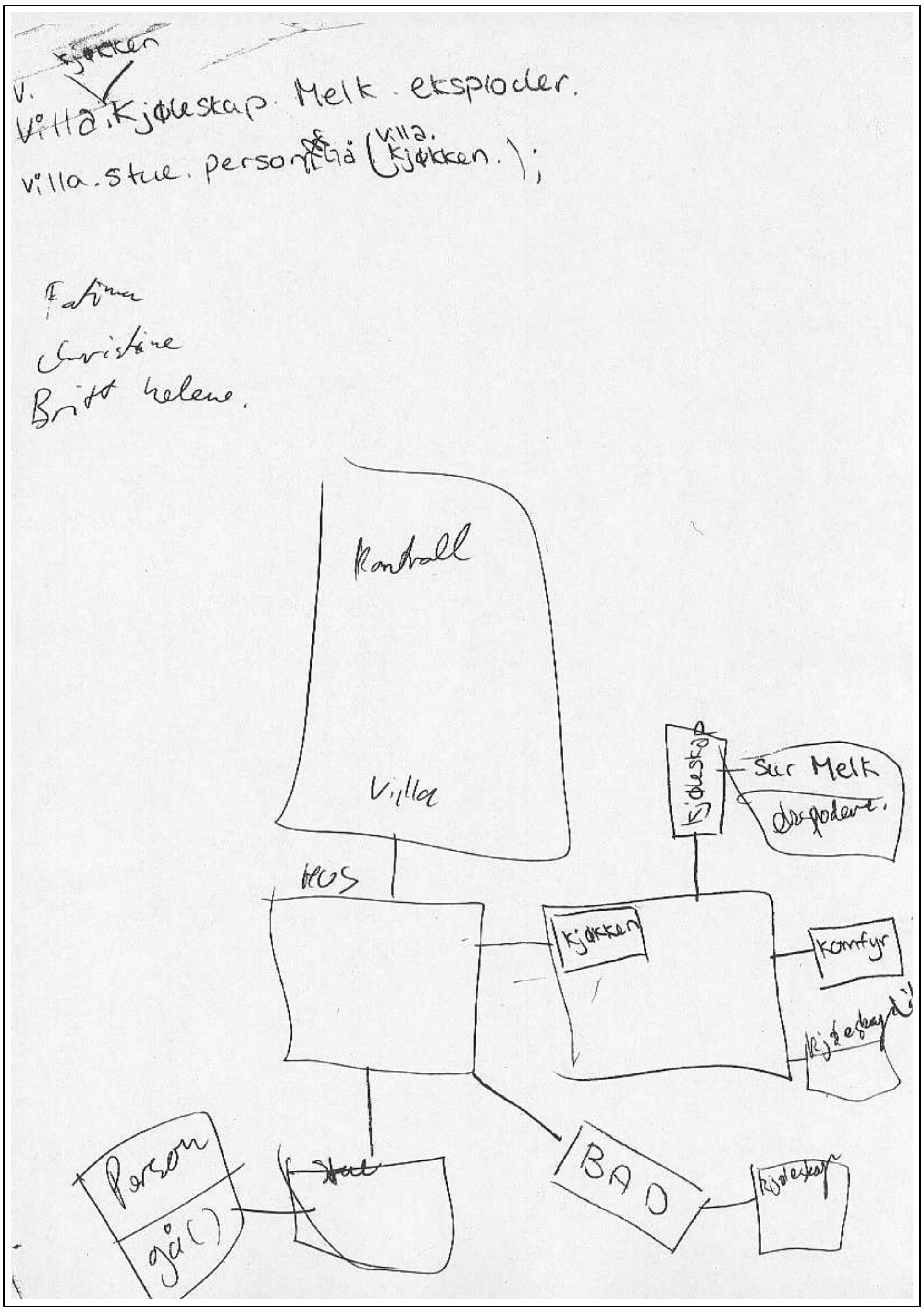


Figure 4.7: A drawing of an object oriented house program, drawn by the children

Chapter 5

Results and Evaluation

This chapter contains the results from both the experiment at Fjellhamar and the experiment at Mølladammen. I have categorized them according to what results are related to programming and object oriented concepts and what results are related to the way children work together when solving this kind of assignment.

5.1 Children working on a difficult assignment

5.1.1 Difficult assignment

At the start of each interview, the last day at Mølladammen, I asked each group how they felt that the exercise had been. Most of the groups were positive. They said that they had had fun, and that they had learned a lot. Most fun was working on a difficult assignment and almost giving up, and then suddenly find an answer to the problem that worked. (Greeno & Collins 1996, page 27) This is very typical in all programming. And it is linked with the behavioristic learning theory. People learn when receiving positive or negative reinforcements. In this case the positive reinforcement was when you finally managed to solve a difficult task after failing many times.

One 14 year old girl thought that it was very fun to learn about how things worked. In this case it was fun to learn about how cogwheels and other parts of the Lego Technics worked. There was also a lot of fun to get a real challenge, but getting stuck on a difficult task was less fun.

During the interviews it became apparent that answering my questions was just as difficult as programming the Lego robot using Java. I know that typing a computer program on a personal computer is something quite different from drawing a UML-like diagram of the design of this computer program. It is also a difference in sitting in a classroom surrounded by class-mates than sitting secluded in a little room with only your group members and a tutor. This change in setting is an example of the situated cognition theory (Greeno & Collins 1996). Learning is a social happening. If the same questions had been asked when the children was seated at their normal place in the classroom, their answers to my question would probably be a lot different from the ones I received.

5.1.2 Program understanding

The children had seen our demonstration-robot follow the black tape on the floor, and had in their heads constructed a hypothesis on how it was doing this. The sensors used polling when checking for change. The reason is described in chapter 3.4.1 on page 30. When programming their robots they tried to follow their hypothesis and if one method of following the black tape did not work, they would try another method. When we asked how they would program the robot to follow the black tape they had different answers. First one would answer:
"We just use the built in function for following a black line"
We would then tell them that no such function exists. Their next answer was much closer to the correct answer:
"The robot will go forward until one of the sensors sees the black tape. Then the robot will turn until none of the sensors sees a black tape"
This is a way of solving the problem, but it is much more complex both for the children and for the tutors, than the idea we had in mind. If the robot should turn and turn until it stopped seeing the black tape, we would have to poll the sensor again an again in very short intervals. Then their next answer was acceptable to us:
"When one of the sensors sees the black tape, the robot turns for a given number of seconds"
This was the way we had solved the problem in our test robot and it was the way we wanted the children to solve the assignment. Then they just had to use trial and error to find the amount of seconds the robot should turn before it continues driving forward.

It was very interesting to see the children somewhat understanding that the first option was difficult, because the robot was very stupid. And then the second option was difficult to implement because the way the robot was programmed did not have any way of "listening" to a sensor. At least not a way we had implemented in our custom made Java API. The reason is explained in chapter 3.4.1 on page 30. The children did not know this, but they only learned to tell the robot to do something for a given set of time, and making it do something until something occurred was not so easy. There was also some "slowness" in the system that made it difficult to "fine-control" the robot So then the children would settle for the third option and be satisfied that they managed to solve the task.

5.1.3 Pair programming

The groups consisted of three people on each group. This usually lead to one of two things. One person doing all the programming while the other two just talked about something else, or two people programming and discussing, and the last person just sitting in the background minding its own business. When interviewing these groups afterward it was very peculiar to find out that these background-people had learned at least as much as the active ones. They were able to participate in the discussions when drawing their robot on a piece of paper. They were also able to correct their classmates when they were wrong.

5.1.4 Shortcuts

A lot of the groups both at Fjellhamar and Mølladammen used a typical hacker-way of programming where the only goal is to get something that produces the correct output. This is the same result that (Kolikant 2004) witnessed in his study. He was studying two people who were working on a synchronization-task. They had gotten an assignment from their teacher that they had to solve. When it produced the correct output, the teacher made them use another set of input. And when their program used this set of input the result was wrong. The people doing the assignment solved the new problem not by going through their code and finding the bug, but rather just exchanged some random semaphore values. After a lot of trail and error the program produced the correct output again and the subjects were satisfied. But the program was still faulty.

One of the groups that did not reach the goal of completing the restaurant assignment was very quick to write a program that produced the correct output, but they had found a flaw in the program and exploited it. I do not think that it was intentional, but they had to write their program in another way nonetheless. One of the problems in the assignment was that when the waiter fetched a course from the chef, he got a random course. So he had to use a method:
findCorrectGuest(objecta.table4)
that returned the guest that had ordered the course the waiter just had gotten. But if you ordered the same course for all the guests, you could skip this method call and just deliver the received course to any guest. And that was what this group had done. This group did not manage to solve the assignment after we told them about their "cheating".

5.1.5 Stoppers and movers

Some of the children are obviously stoppers while others are movers and some are the kind of movers that just keeps patching their program over and over again (Robins, Rountree & Rountree 2003). Some of the children give up at the first sign of trouble and put their hand in the air and asking for help. And while waiting they do not even look at their problem. Others put a hand in the air when getting stuck, but found the error themselves before one of the tutors managed to help them. And a few of the children were very keen on finishing the assignment at any cost and just tried and failed over and over again, without really trying to find their original error.

5.2 Age and gender

5.2.1 Age

Children are trying to find out who they are, and at the same time trying to be special. This was easy to see when the 11 year olds were building their houses. They were always looking at each others houses, and did not dare to be very original. Except for some few of course, that wanted all the attention they could get. Compared to the 11 year olds, the 14 year olds used more energy on getting recognition from the other people in the classroom, trying to act cool and saying funny stuff.

Most of the 11 year old children thought it was most fun creating Lego-robots that could move. The children at Mølladammen thought it was less exciting to build the Lego robots and more time was spent programming the robot instead.

At Fjellhamar with the 11 year olds, a lot of the time was used getting to know Lego Technics and designing a sturdy robot. And approx half of the groups spent most of their time building fancy robots and using the pre-built programs just to make the robot go forward. At Mølladammen we had made a template for a very sturdy robot that the children could follow. So the 14 year olds spent considerable less time trying to build a cool and fancy robot before using the template at built the sturdy robot we had planned that they should built.

Compared to the 11 year old children at Fjellhamar, the 14 year olds did not have any more experience using Lego or Lego Mindstorms. Children stop playing with Lego at a certain age. They were not so eager to start with the building once they got permission to open their Lego boxes, and they did not build the same kind of imaginary Lego vehicles as the 11 year olds did. Because of this I think that the 14 year olds felt that they were a little too old to play with Lego.

The 14 year olds did not have any more experience using the Technics side of Lego than the 11 year olds had. They had no more understanding of how to use cogs and shafts either. And the same can be said about their knowledge of general physics. I asked a random group if they could explain to me why tires with rubber "stuck" more to the floor than ordinary hard Lego bricks. I wanted them to explain to me about the friction. But even thought they had a general feeling, they were unable to explain it to me.

14 year old children do not have much physics in their curriculum, so it is understandable that they had problems explaining friction to me.

5.2.2 Gender

We found that 14 year old children, especially the girls, had very little interest in programming Lego robots using Java. But the girls still wanted to complete the project, so that they could say to the other schoolmates that they were done. They did what they were told and tried to find the easiest way to get the program completed and working. At Fjellhamar with the 11 year olds the difference in interest between boys and girls was not that noticeable.

During my interviews the last day at Mølladammen I found that the girls were no worse at answering my questions, but they cited the API instead of telling me what they thought. For most of the girls this was "just another school exercise".

At Mølladammen the girls had very strong personalities, and the boys were more in the background. Some of the boys enjoyed programming, and tried their best to program a good Robot, but they did not seem as smart of fast as the girls. In a group consisting of two girls and a boy, the boy would try hard to understand the program, and the girls would just chat about other stuff and help occasionally when the boy was stuck. Girls in the early teens often seem smarter than boys at the same age, and I guess that is because they mature earlier.

When we removed the Lego from children at Mølladammen the last day the feelings were mixed. We got the feeling that some of the children, especially the girls were tired of Lego and of programming. This is understandable, since it was very demanding at times. And they were also 14 and Lego is not cool with 14 year old kids. One day we had a case of a heartbroken girl, and this really devastated the whole class. This was a very strange experience, to see a whole classroom fall apart because of one girl.

During the interview a boy said that he found the experiment fun a first, but it became a little repetitive on the last day. He hoped that they could put more gadgets on the robot, for example a web-cam. This is typical boyish behavior. They usually appreciate technical gadgets, and would appreciate that the robot could do all sorts of things.

At the start of each interview I wanted to know the name the group had chosen for their robot. In one group they had the same name both on their robot and on their Java-file. One group even said the name of their program file when asked the name of their robot. This group only consisted of girls. All the other groups were mixed. I am not sure why they think that the name of their program is the name of their robot. And when asked to describe what they did with their Lego-robot they answered that they solved assignment 1 and 2. Again this shows a typical way of behaving typical for most of the girls participating in the exercise. They have no real interest in the project, but since it is a school-project they have to solve the assignments given to them.

One group, when asked to describe what the robot consisted of, misunderstood and thought I wanted them to explain how the physical robot worked. The two girls on that particular group pushed the boy to do this explaining. But when they found out that it was the robot-program that should be explained, they were at least as active as the boy. This shows that the girls were interested in the assignment since it was schoolwork, and you are supposed to complete your schoolwork. The boys seemed to enjoy the actual programming and Lego-building.

5.3 Declaring and assigning pointers

We did not use much time teaching the children about declaration and creation of new objects. At the top in their robot class they had to declare their object:
LegoMotor leftMotor,
and then again inside the constructor they had to make the new object:
leftMotor = new LegoMotor(A)
We did not tell them much about the constructor, just that it was something that had to be there. They often forgot to write either the declaration on the top of the robot-class, or the assignment in the constructor.

This leads me to believe that they had problems understanding the difference between the declaration and the assignment. The children usually copied and pasted from previous code and did not bother to try and understand why they wrote either of them. If we could have avoided this confusion we would have, but declaring the correct reference to a objects at the right places in a Java program is still a big part of object-oriented programming.

During an interview, when the children drew a gardener-object (see figure 5.2 on page 65), they called the class itself Torleif (with a capital T) and not Gardener. A lot of the participants of this project had that same problem, to see the different between a type of object (class) and the name of the object (pointer). This is probably because of the strange way an object is created in Java. When you want to create an object of a class you have to use two lines of code to declare the object. First you have to declare what kind of object the pointer is, and then you have to assign the new object to this pointer. This is what is done at the top of this chapter. There is a shorter way to do it as well, but we did not use that in this assignment. It is pretty hard to understand why the declaration have to be in one part of the program, and the assignment in another part of the program. Especially when you do not really know what declaration and assignment are. At last the gardener is placed in the garden, and is given a method called *cleanup(milliseconds).*

Using a method that returned an integer was something the children had used when they programmed the Lego robot. In the restaurant assignment they had to use a method that returned an object, a person object. They had to use this method when the waiter had fetched a course from the chef and wanted to deliver it to one of the guests. This could be done in two ways. Both the ways were used by the groups that managed to solve the assignment.

1. By writing a code line that used both the deliverFoodCourse and find-CorrectGuest methods like this:
   objecta.table4.waiter.deliverFoodCourse
   (objecta.table4.waiter.findCorrectGuest(objecta.table4));

2. By using a temporary variable to store the correct guest and then delivering the correct course to this guest like this:
   guest = objecta.waiter.findCorrectGuest(objecta.table4);
   objecta.waiter.deliverFoodCourse(guest);

The second way, they way with the temporary variable shows that they have an understanding of what an object is and how it is treated. And the difference from this person object and i.e. a LegoMotor object, is that the person object does not have a physical representation. In the real world the waiter would probably have gotten a sheet of paper with the name or reference number of the person receiving the course, and not the whole person or a reference to the person. Or maybe the sheet of paper with the name of the recipient can act as a unique identifier and as a reference.

5.4 Class vs. Object

In the object-oriented paradigm it is often difficult for beginner students to understand the difference between an object and a class, especially if they only create one object of a class.

The call
murderer.lilleputt.kill.garden1.swing.child
. has a lot of errors. The murderer named lilleputt is supposed to kill the child that is using a swing in the garden. The start of the call
murderer.lilleputt
contains both the class murderer and the reference to the murderer object called lilleputt. This is an indication of the difficulties of understanding the difference between an object and a class. The rest of the errors of the statement is explained in chapter 5.7.1 on page 75

During an interview I asked the group if it was possible to have more than one robot. The answer was yes, but they could not have the same

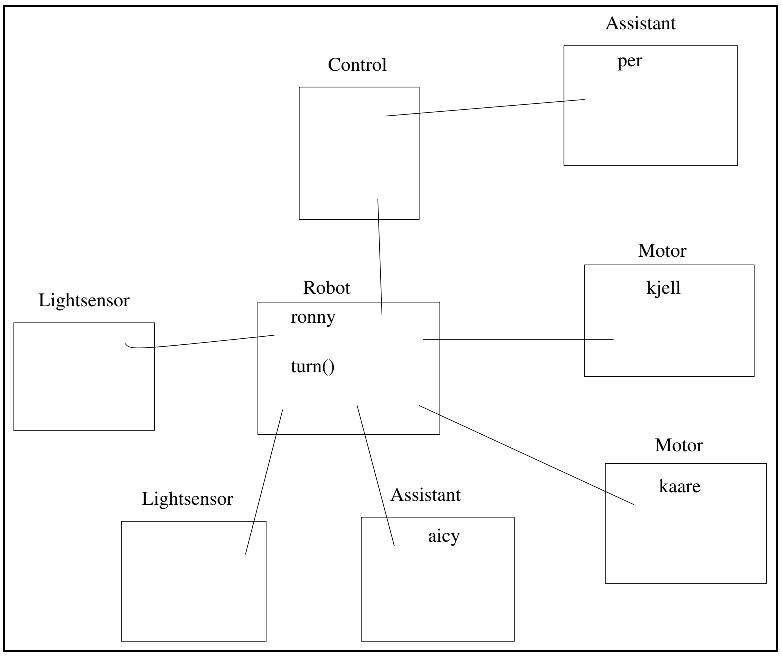


Figure 5.1: Drawing of a robot using an UML-like notation

name. The children and the teachers use the term “name” as the name of the pointer to the actual robot-object, and not a name-variable in the robot-object. They knew about the control-class as the “start” of the program, which was described as a control-center for controlling the world. Much like a main-method, but without the Java static-problems. This is visible on figure 5.1 as well, where the control-class does not have a name, and does not have a pointer to it from anywhere.

After talking about the Lego-robot and drawing their Robot, they were asked to draw an object-oriented house in the same way they created their robot. They were given the same start material as with the robot. First a control-object, as a remote-control to the different parts of the house. This was because it should look the same way as the Lego-robot. In the control-object there was a pointer, named “hansen” to a house-object. Exactly in the same way as the control-object had a pointer to the Lego-robot. One student asked if they needed to call this pointer to the Lego-robot object something. This question may show that the children have learned that an object is something you need a handle in order to use.

Figure 5.2 on the following page shows a translated and redrawn version of an object-oriented house drawn by one group during an interview. And looking at the drawing the children have to tell the gardener to cleanup for a number of seconds. The method is called from the control-object. The boy on the group’s first try at the long call sentence was:

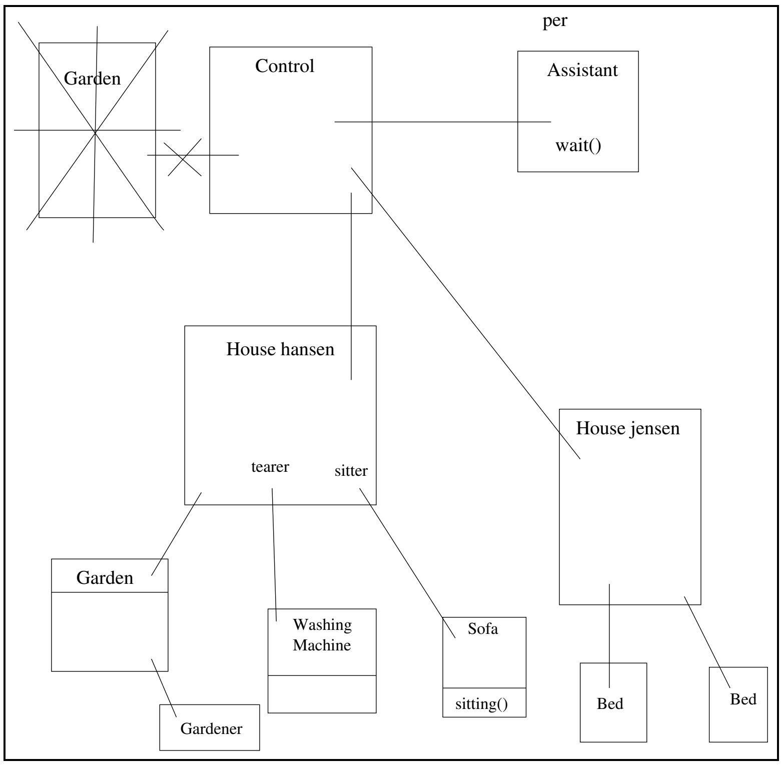


Figure 5.2: Drawing of an object-oriented house using an UML-like notation

control.househansen.garden.gardener.cleanup(1000)
This call has a number of errors, but some of them are because of my inconsistence when I helped them to draw their house. To start with you do not have to type control when you type the call from the control-object. This was not done so much in the coding, but when they see a visual representation, it is easier to see that they have a control-object, and that it all starts in this object. The error househansen is caused by an inconsistent way of drawing the different objects of the house(see figure 5.2). We had given the gardener a name, so he should have used this name instead of just writing "gardener". To me it seems that it was difficult, for the children, to see the difference between the type(class) and the name of the object or the pointer to the object.

When asked to attach the motors to the drawing, a girl made a box with the name Motor over it and the names kjell and kaare inside. Kjell and kaare were the names they used on the motors on their Java-program. When asked if they really made a motor called "kjell kaare", the answer was no. Then she made another Motor box and called it kaare. But still when she should place the light-Sensors on the drawing, she only made one box. It seems that this girl had difficulties understanding that they had made two motor and two sensor objects in their computer program. The reason is probably that the two motor objects were of the same type(class). The two sensor objects were also of the same type and caused the same confusion. This is another example of the difficulties in understanding the difference between an object and a class.

During the interviews the groups were given random tasks to complete. This was mainly because I had very few questions prepared, and I did not always remember what I asked the other groups about. One group were asked to place a sofa in the house. This task was a little difficult, since they did not have anything to look at. There was no possibility to copy-paste here, so they had to make the transfer from what they learned about the robot, the attached objects, and the house. After some thinking, the active boy of the group got an idea, and drew a little box called sofa beside the house-box, and named the new box: Sofa. The same boy said that this sofa had the property: "you can sit in it". Another question asked was: "If we wanted two different sofas with different properties, could we call them both a sofa?". They answered the question correct with a negative answer. If two objects have different properties, they are different types of objects.

In order to make something that was not directly the same as the robot-program, I asked one of the groups to place another house in their program. And they did correctly by making another square beside their first house-square and placing a pointer in the control-object named something else than the first house. This was a little unexpected, since they had not done anything like that before, and they had not even thought in the same manner when they always just had one object in the control object (except the assistant). I draw the conclusion that they had learned something about how to be able to control an object that they made.

5.5 Dot-notation

Understanding how the different objects interacts and how they are interconnected is part of what you have to understand in order to use Java and object oriented programming. How the children understand and misunderstand the use of this dot-notation will give an indication of how much they understand. It will also give an indication for what is difficult and not.

There were two different ways to talk when the children called the methods. Along with their writing they said either "period" or used the genitive "'s". Figure 5.3 on page 68 shows a discussion between a girl and a boy where the boy says "period" instead of using the genitive s.

The people that said "'s" as in "volvo's leftMotor's forward" showed a little more dotting-understanding in what they really were doing. By dotting-understanding I mean understanding where to type the dot when writing statements that access a specific method at a specific object. I.e. a statement

like:
volvo.display.write("Hello World");

Figure 5.4 on page 69 is an example of the lack of understanding of where to apply the dot when using dot-notation. It shows the discussion on how to change the number of seats. It also shows the difficulty of understanding that in a parameter you also have to say where the object is located, just as in the discussion in figure 5.9 on page 76.

5.6 Program design

With the use of control technology the children had the possibility to see what their programming actually did. The physical Lego robot would be a real world object of the robot class they wrote in Java, and the Lego robot would behave according to the programming in the control class.

When the correlation between the programmed robot class and the physical robot object was more or less one to one, it might be easier for the children to understand the reasons for their programming. Figure 5.5 on page 70 and figure 5.6 on page 70 shows this correlation between the physical robot and the program code.

Because of this one to one correlation we hoped the children would be able to place the different Lego objects in the correct class. That they would place a new LegoMotor in the Robot class and not in the control class. The same goes for the different sensors, the LCD display and the Speaker. And when the children would write their controlling code in the control-class they would have to access the motor and sensors through the robot object using dot-notation.

The discussion in figure 5.6 on page 71, from the Lego Robot programming, shows how the children did not think in the manner we wanted them to and only copied what they had already written. In this scenario the Speaker object is incorrectly placed in the control class.

And the children ended up placing the Display at the same place they had placed the Speaker. In the same groups' program the motors and the sensors were placed correctly in the Robot class. So it was strange that the Speaker and the Display would be placed in the control class. My theory is that since they did not physically place the Speaker and the Display on the robot in the same way they did with the sensors and motors, they did the same thing with the Speaker and the Display that they had done with the Assistant. And that was placing it at the top of their control-class. Figure 5.8 on page 72 shows a part of their code. It does make the robot do what they wanted it to do. But we wanted them to place the Speaker and the Display in the robot class instead. I would want to say that they did not think correctly, but what I mean is that they did not think in the manner we wanted them to.

This conversation starts when the group has made an if test that checks if one of the light sensors sees a black tape. They return from the testing and starts writing the test for the other one.

```
boy:   ``come, we have to write another one.'' (Another if test)
girl1: ``another what?''
girl1: ``Hey, I was sitting there. I want to write.''
boy:   ``I do not care.''
boy:   ``if....'' (typing if( under the previous if sentence)
girl1: ``but I wanted to write.''
boy:   ``volvo period mummi period luminosity 45''
       (writes volvo.mummi.luminosity)
boy:   ``is there any capital letters in luminosity?''
girl1: ``no''
boy:   ``paranthesis''
girl1: ``out''
boy:   he types types ()
girl1: ``now you need a bigger than, smaller than, character''
girl1: grabs the keyboard and types <45)
boy:   the boy grabs the keyboard again. He then starts to write: volvo
girl1: the girl starts building with the Lego bricks and not paying
       attention.
boy:   ``Shit, it is written too far away''
boy:   he then deletes the volvo characters he has writting and looks
       at the girl, obviouly wanting some help.
boy:   after a little while he sees that he is missing the curly braces
       that needs to be placed after the if clause
boy:   he looks all over the keyboard but does not find the character
girl2: the other girl on the group arrives and looks at the monitor
girl2: ``did not it work? Why are you doing it again''?
girl1: ``we have to do it for the other light sensor aswell''
girl2: ``aha''
boy:   the boy find the curly braces and types it in.
girl2: leaves again
girl1: now pays attention to the boy
girl1: ``leftMotor stop'' She does not say period between the words
boy:   the boy types volvo.leftMotor.stop()
girl1: is not paying attention again
boy:   types a copy of the previous if sentence
```

Figure 5.3: A conversation using “period” instead of ’s

```
girl:   ‘‘we have to change the number of seats at the table’’
tutor:  ‘‘yes’’
girl:   ‘‘is that the butler?’’
tutor:  ‘‘yes it is the butler, the butler of the restaurant objecta’’
girl:   ‘‘so then it is objecta period butler period changeNumberOfSeats’’
        while talking she types: objecta.butler.changeNumberOfSeats(
tutor:  ‘‘what does the API say? What table to change..’’
girl:   ‘‘table four and number of seats four’’
        she writes (4, 4)
tutur:  ‘‘yes...but what was the name of the table?’’
girl:   ‘‘the tables’ name was four’’
boy:    ‘‘table four’’
tutor:  ‘‘yes...table four is correct’’
        ‘‘but where does the table belong?’’
girl:   ‘‘objecta period table period four’’
tutor:  ‘‘uhm...yes, it is objecta’s table four’’
girl:   ‘‘so there is no period between table and four?’’
boy:    ‘‘no’’
```

Figure 5.4: Difficulty using the dot notation

During the interviews I wanted the groups to explain to me how their robot-program worked, and how this correlated with the physical Lego-robot that they had built. They got a piece of paper and a pen. My plan was to make them draw a UML-like diagram. These kids did not have any experience whatsoever with that kind of thing, they had just written their first computer program two days ago. My plan was that if these children managed to draw their program on a piece of paper, they would have an overview, and maybe understand a lot. Figure 5.1 on page 64 is a translated version of one of these drawings.

After drawing their robot on the paper I wanted to find out whether they understood why they had drawn their robot in this way. How they could use the names(pointers) of an object to issue commands to either that object or to another object even deeper inside the structure. And if they got the dot-notation correct, they would understand some of the program structure. In order to reach the robot, one child said in a very insecure way, that you had to type *control.robot1.....* and then mumbles something incoherent. This was said while looking at the UML-like drawing they just had drawn. The statement is wrong because you type the statement in the control class and therefore do not know about any control-reference. If control at the start of the statement had been removed it would have been correct. But still this shows that the UML-drawing is being used when trying to give the correct answer, because while the child was formulating the desired statement her

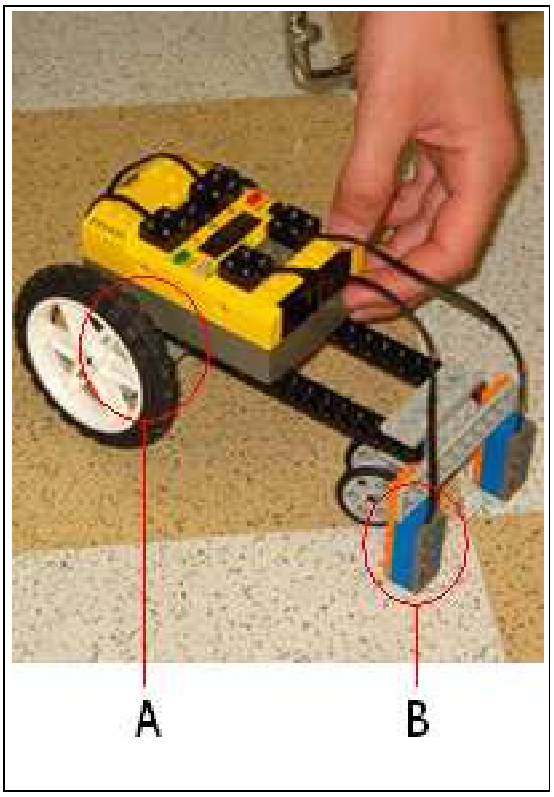


Figure 5.5: A physical Lego robot with a motor and a sensor

```
class Robot{
  Lightsensor rightLight;
  Legomotor rightMotor;

         Robot(){
```

//rightLight is sensor B on figure 5.5

```
     rightLight = new Lightsensor(1);
```

//rightMotor is motor A on figure 5.5

```
     leftMotor = new Legomotor(A);
  }
}
```

Figure 5.6: Java representation of a Lego robot with a motor and a sensor

```
A child raises her hand and asks the tutor how
she can make the robot write something on the display:
Child1: ``How can we make it write name?''
Tutor:  ``Write?''
Child1: ``Name''
Child2: ``I want to write something on the display''
Tutor: ``Aha, write on the display''

The tutor then uses their printout of the API and tells the children
to use Display.

Child3: ``Class Display''
Tutor:  ``You have to find out where the Display belongs''
Child3: ``Is it here?'' points the mouse to the top of the Robot class
Tutor:  ``Where does the Display belong....it belongs to the yellow box..so....''
Child3: ``Ok...here then'' and points to the top of the Control class.

At the top of their control class they have already declared and made
a new Speaker object. This is not correct, since the speaker also is
a part of the robot and not of the control.

Tutor:  ``Sounds good, it was here you made the Speaker''
Child2: ``What shall we call it?''
Tutor:  ``You have to make a new Display first''
Child3: ``How do we do that?''
Tutor:  ``Look at how you did it with the Speaker''
```

Figure 5.7: Placing of the Display object

```
Robot volvo;
Assistant per;
Speaker lars;
Display beate;

  Control(){

  volvo = new Robot();
  per = new Assistant();
  lars = new Speaker();
  beate = new Display();

  lars.playTone(738, 50);
  volvo.leftMotor.backward(2);
  volvo.rightMotor.forward(3);
  per.wait(2000);
  lars.playTone(440, 70);
  lars.playTone(536, 30);
  lars.playTone(694, 48);
  volvo.rightMotor.forward(5);
  lars.playTone(559, 50);
  volvo.turn();
  beate.write("Hello");

}
```

Figure 5.8: A working but faulty control class

finger first moved to the control class and then over to the robot class.

One group had an assistant both in the control-object and the robot-object. I wanted to see if they understood the program structure and how the references worked. So I asked them if the control object could access the assistant in the robot object, and if the robot object could access the assistant in the control object. The answer was that control could get hold of the assistant in the robot-object, but the robot could not get hold of the assistant in the control-object. I am not sure why they managed to answer this one correctly. I did not ask why, mainly because I was so surprised of the correct answer. They could have been lucky with their answer, or they could have tried it in their own program. They may have thought that the lines on the drawing are one-way only. Figure 5.1 on page 64 is a translated version of the figure this group had drawn, and this figure does not say anything about the lines being one way only.

Not many groups created their own methods. Figure 5.1 on page 64 shows that one group created a turn method and placed it in the robot class, but still their control-class became very big and difficult to get an overview of. Often the students lost track of their program, and did not understand why the robot acted in the manner it did. It was also difficult for the teachers to help the children understand their program when it consisted of 20 or more lines of commands for the robot, and no structure at all. In the example described in figure 5.6 on page 71 the tutor misunderstands the program and does not notice that they are making Display in the control-class.

One group was asked what the robot consisted of, and asked to draw this on the paper. At first one on the group draw a motor-object and attached it to the robot-object. When asked if the robot only consisted of one motor, another on the group drew the second motor on the drawing. It also consisted of two touch-sensors and one light-sensor. The interesting thing about these sensors was that when the children drew them on the piece of paper, they did not make object-boxes like they did with the motors, but instead just wrote the sensors directly into the robot-object. The physical motors are a little bigger and are placed on the A-B-C ports on the RCX instead of the 1-2-3 ports. But since none of the other groups did the same "mistake" this was probably just a coincidence.

When one group was asked what the robot consisted of they mentioned the methods *driveForward()* and *driveBackward()*. This was not something that we had written in their template, and they had not made any methods in the robot that drove it forward and backward. When asked how they got the robot to drive forward, they remembered that they had to write *volvo.rightMotor.forward()*. It was not obvious that they had to attach motors to the robot (in their drawing) before they would be able to make it go forward. Maybe they did not really understand that the robot consisted of different parts like motors and sensors. And since their template had the line *volvo.rightMotor.forward()* they just had to copy and paste this line, and did not have to understand why they wrote what they wrote. After a little discussion with me, they agreed to that there was no such thing as a *driveForward()* method in the robot.

During the object oriented house part of the interview, one of the groups were asked to draw a living room in the house. One of the boys on the group asked if they had to make a room first. I feel that this is very object-oriented thinking, but probably a coincidence, since the programming of the Lego-robots had nothing to do with this kind of thinking. I did not give the students a definite answer to this question. Instead I asked them about the different rooms in the house. Why did we call one of the rooms kitchen, and the other one a living room, when obviously they were both rooms. After this little discussion they wanted all their rooms to be of the type Room. So both the kitchen and the living room were kinds of Room, then only thing different between them was the names of the pointers. But since one of these rooms has a pointer to an air-conditioning-system they have to be different in some way. There was also some discussion about whether the air-condition should be in the house-object or in the kitchen object. They agree to place it in the kitchen, because it is there it is needed.

This could have been explained if we had introduced subclasses to the children. But since we had not mentioned subclasses during the programming, I did not want to confuse the children with another concept.

5.7 Method call

During an interview one group of children were asked if they could write *per.forward()* (per was the name of the assistant). They all agreed that they could not. The reason was that only a motor could go forward and backward, and per was just an assistant. So obviously it was easier to understand that the assistant was unable to go forward compared to what the robot was able to do. In the course we had not used much time explaining how to create your own methods, and how to use them. Some groups had created their own methods anyway, usually with lots of help from the instructors. One group had made a method *turn()* in the robot, and were using this from the control-object. And when calling the motors from this method in the robot-object they did not have to write *volvo.rightmotor.forward()*, just *rightMotor.forward()*. I am not convinced they understood why they wrote the method. But afterward when I interviewed them and they were able to see their drawing of the robot, it got clearer why they did not have to tell who's rightMotor to access, since they already were in the correct robot-object. Still, this was not the norm. More than half the children interviewed answered incorrect when they had to make a call they had not made before. Like calling the motors forward-method from the robot. The answer was usually the same as when it should be called from the control-object. Obviously they had problems to see how the "world" works from inside the robot object compared to outside the robot object in the control object.

During an interview, one group made an air-conditioner that could blow hot air or cold air. When asked how to make the air-conditioner blow hot air, the one of the boys answered correctly: *ville.kitchen.dc.blow(hot).* He also manages to explain why in a satisfactory way. Then a thief called staale shows up in the control-object. This thief wants to steal the air-conditioner. The girl that memorized the API answered more or less correctly: *staale.steal(ville.kitchen.ac).* But when asked if the thief would steal both the house and the kitchen, she said "maybe it would?". This could be because she thinks that a parameter can consist of many inputs at the same time, or it could simply be because she is very uncertain of what she is doing, and is only copying something that she remembers, but have not understood. The boys of the same group are pretty certain that the thief only will steal the air-conditioner, because when asked one of them said that the thief would steal the ac that belonged to the kitchen in the house.

5.7.1 Parameter problem

On one group we made a public garden which was not part of a house. And in this garden we made a swing, and a little child using the swing. This child had one method. To *swing()*. And then we placed a murderer in the control-object. This murderer could *kill(person)*. And I wanted the children to kill the little child swinging in the garden. This murderer is called lilleputt and is of the class Murderer. I ask the children what to type in order for the murderer to kill the swinging child. Their first try was:
murderer.lilleputt.kill.garden1.swing.child
. This was not entirely correct, but I could see they got the basic idea. Our course did not teach the children about parameters in any good way, so very few of the children interviewed got this correct. And during the interviews I was very inconsistent about giving the different objects names. So in this case murderer and the garden had gotten a name, but the child and the swing had not. Since there were only one swing, we had not given the children any reason to name it in order to get at it. The beginning of the command:
murderer.lilleputt
is also wrong. The explanation is given in chapter 5.4 on page 63

It contains both the class murderer and the reference to the murderer object called lilleputt. It it clear that it is difficult to understand the difference between an object and a class.

During the restaurant assignment, one of the groups reached their hands in the air and asked a tutor how they were supposed to use the menu object. In order to explain it, the tutor pointed to another error in their code. The group had written the following line in their control class code:
objecta.butler.changeNumberOfSeats(table4)
The correct answer would have been:

objecta.butler.changeNumberOfSeats(objecta.table4)
The reason is that table4 is part of the restaurant object and not part of the control class. Figure 5.9 on the following page shows the discussion between the group and a tutor trying to explain their error.

From this conversation I find that they found the error themselves, though with very much help from the tutor. But still on the calls for seating a person at a table:
objecta.butler.placeByTable(person, objecta.table2)
they still forgot the objecta in front of the table. They struggled with the new error at least 10 minutes even thought it was the same cause as before. It is understandable that reading and understanding compile-errors is difficult, but even with trial and error they were not able to see that this table

```
tutor:  ``why have you written: objecta.butler ?''
girl1:  ``because we thought that.....''
tutor:  ``I am not saying that what you have written is wrong''
girl2:  ``because it is the butler to the....''
girl1:  ``because it is the butler at the restaurant objecta''
tutor:  ``correct. Table4, where is that located?''
girl1:  ``at the restaurant objecta''
tutor:  ``so the butler belongs to the restaurant and...''
girl2:  ``and it is the restaurant's menu''
tutor:  ``yes, but as you said, table4 is located in the restaurant''
girls:  ``???''
tutor:  ``you cannot write table4, it needs something more''
girl1:  ``no..''
tutor:  ``yes..''
girl1:  ``aha, so you have to type objecta here''
        points to the start of table4 and types: objecta.table4
tutor:  ``yes, you have to spoonfeed the computer''
```

Figure 5.9: Trying to get a group to find their own mistake

was the same table as the one in objecta.

Failing to write the whole reference to an object like the the example above was more frequent inside parameter parenthesis. The example in figure 5.9 shows that the children understood why they had to tell the computer that it was the butler in the restaurant that should do something. But they did not understand that it was the table in the restaurant that the butler should do something with.

The reason for this difficulty in using the dot-notation in a parameter between two parentheses could be the following. When programming the robot and using methods with parameters, they never used any kind of dot-notation. Either it was *forward(5)* or *write("something")*. It could be that the children did not generalize enough. And since they never made a method themselves that used parameters, I would think that they did not get a real understanding for what a parameter was.

5.8 Problems with Java syntax and semantics

A big problem with Java is the syntax and the semantics. My own experience in Java programming have shown me that the use of some characters are difficult. One thing is to remember to place the semi-colon at the end of each statement. This is something I forget to do all the time, and when I compile the program afterward I sometimes get an error message in another

```
[ ]   -   brackets
( )   -   normal paranthesis
{ }   -   curly braces
```

Figure 5.10: Different characters that look the same

part of the program because of the missing semi-colon. The same goes for using parenthesis, brackets and curly braces. It is very easy to use one of them instead of the other.

The children often used the wrong kind of parenthesis in their code, and when compiling the program they got an error they did not understand. When they put a hand in the air and one of the tutors tried to help and find the error, they also found it difficult to find the parenthesis that was wrong. This is a problem that a lot of the editors that is used when writing program code. The characters in figure 5.10 is very difficult to distinguish from each other. But as you can see in the figure, this is not really a problem when the characters are printed out on a piece of paper.

This problem with the use of different parentheses was often solved by trying and failing. The children relied mostly on copy and paste from already generated algorithms.

There were some troubles commenting out pieces of the code. In Java this is not always as easy as it should be. Commenting in Java can be done in two ways. Either you use two slashes at the beginning of the line you wish to comment, or you can comment out a larger part of the code by using slash and asterisk at the start, and the opposite, asterisk and slash at the end. The problem with the Java way of commenting code is that it is difficult to see where the comment ends, and if you have lots of comments all over the code it can be very hard to keep track of where the comments starts and where it ends.

Another problem some of the groups had, was to find the correct character on the keyboard. In the discussion in figure 5.3 on page 68 you can see how they boy has problems finding out how to type the curly braces. The parenthesis was easier because they had used this in other programs, and you get to it by using the shift-key on the keyboard. More difficult was the curly braces since they were not a “normal” character, and it was just more difficult to type, But I think that using the “alt gr” key on the keyboard is easier for norwegian children than it would have been for english children. The reason for this is that in order to type @ on a norwegian keyboard, something youth do all the time, you have to use the “alt gr” key instead of the shift-key which is used on the english keyboard layout.

5.9 Real World

Since we had pushed the term "properties" so much during the experiment, I asked the children the difference between a motor and a sensor. All the students answered correctly, that the properties were different. And this was the answer I was looking for.

The group consisting of girls wanted to decorate the kitchen in order to make the room into a kitchen. A room without correct kitchen-wear was not a kitchen. This is a very natural way of thinking, but has little to do with programming and naming classes. They got to put a refrigerator and a range in the kitchen. I asked them what the range and the refrigerator had in common, and the answer was that they both belonged in the kitchen. And this is true enough, though it has little to do with their properties.

The same group had a lot of problems when I wanted them to tell me how different call-statements from the control object should be written. I wanted them to make the milk in the refrigerator explode. The milk had a explode method, so they only had to call this method in the correct manner. Their first try was
refrigerator.milk.explode()
I did not bother to get the children to name their object, so the object-type and the object was often the same. Anyway this was not a statement that would work, since the statement should be written in the control object. Their next attempt at the same statement was
ville.refrigerator.milk.explode()
This answer was not correct either, even if they remembered the house ville. They still forgot the kitchen. Maybe they would have been more successful if they had looked at their UML-like drawing while giving me the answer.

Their next task was even more difficult. I wanted the person in the living room to go to the kitchen. And the called should be made from the control-object this time as well. They started correctly this time around:
ville.livingRoom.person.walk(kitchen),
but it was not correct. I tried to explain that the control-object did not know about the kitchen, and neither did the person in the living room. They feel that this is very strange, since he lives in the house. Logically he should know about the kitchen and be able to go there without much difficulties. Since they had not done this kind of calling in their Lego-programming I did not expect them to succeed. But they clearly have a hard time seeing how the world looks from the control-object. (Pea 1986) talks about a superbug. He says:

> The superbug may be described as the idea that there is a *hidden mind* somewhere in the programming language that has intelligent, interpretive powers. It knows what has happened or will happen in lines of the program other than the line being

> executed; it can benevolently go beyond the information given to help the student achieve he goals in writing the program.

And this is clearly what happens here as well. Since the person lives in the house, the computer (or this person), knows about the other rooms in the house and does not need to be told where the kitchen is located.

One group thought it was unnecessary for the waiter to follow the guests to their table one by one like this:

objecta.butler.placeByTable(person1, objecta.table4);
objecta.butler.placeByTable(person2, objecta.table4);
objecta.butler.placeByTable(person3, objecta.table4);
objecta.butler.placeByTable(person4, objecta.table4);

and rather wrote:

objecta.buther.placeByTable
(person1, person2, person3, person4, objecta.table4);

While writing the call above, the children said that it probably would not work, but they did not want to fix it just yet and continued with the rest of the program. This example shows that they are trying to use the real world analogy of a butler seating the guests at a table. In the real world the butler would have seated all the guests at once, in the same manner as they did in their program. But our API did not support this way of seating the guests, so when they tried their program they got an error, and were quick to fix it by seating one person at a time instead. Seeing how their first try is much more like the real world, we would probably support that way of seating the people as well if we wanted to make a new version of the restaurant simulator program.

When writing the next part of their program, this group wondered if they had to write a command line for each of the guests when selecting from the menu. In the real world the guests would each take turns and placing their order at the butler. And this was what the group did in their code as well. So they had four calls like this one:

person.chooseFromMenu(3, 2):

where the first number indicated the number of the main course and the second number was the dessert.

During an interview when the children put two beds into the house, they wondered if they could compare these beds with the motors on the robot. And I find that this is a very good comparison. But when the house also needed a gardener the children had different opinions. One alternative was to have the gardener in the control-object, but the others on the group felt that it would not belong to the house if they did it that way. And this is some of the object-oriented encapsulation that we wanted to teach the children in this experiment. To put the different objects where they belong, and not where it is easiest to access them. But they did not see the point of the house having a gardener, when it did not have a garden. So a garden object was made. But where should the pointer to the garden object be placed? One of the girls meant that it should be in the control-object since it would not fit inside the house because of size. And when every other object we have talked about until now always fits inside or on top of the other object, this was a logical way to think. The others on the group did not agree, and said that the garden still was a part of the house and the pointer should therefore be in the house. Another reason was that if the garden was placed in the control-object it would be more like a house of it own. This last argument shows that the placement of an object has a lot to say about its properties. I.e. that a garden placed in a house is smaller that a garden placed outside (in the control-object). Compare a public park with a private garden.

A group made a garden and a gardener in their object oriented house. When writing a call with the gardener, the two girls on the group felt that the call-sentence was unnecessary long. They claimed that since the garden is such a big part of the house, the program already knows that the gardener is located in the garden. I wonder if it is the size of the garden that makes it so obvious that the gardener should be in the garden, or if it is the fact that the gardener is a gardener. Obviously there is a big difference in the way the garden and the sofa are a part of the house.

Another issue with the garden is how to get to it. Since you can not get to a sofa unless you enter the house, it is natural that you have to go by the house-object in order to obtain the sofa-object. But the problem the garden causes is that this analogy will not work anymore. Usually you can get into the garden without first having to go into the house. Sometimes you even have to go through the garden in order to enter a house. And this causes a lot of confusion. Several of the students meant that the garden was not part of the house.

The children might have thought it was logical that the motors should be placed in the robot-object and not in the control-object. The reason I think they might have thought that was that they said that it "belonged" there. And this was just what we wanted the children to see. Even if it is easier to put the motor in the control class, in order to access it directly, it is not the way we wanted them to do it. We wanted them to think of how it was done in the real world, where the motor is a part of the vehicle. A beautiful example of encapsulation and responsibility-sharing. And I think they did this correctly because they physically placed the Lego-motor on the Lego-robot. It was not that obvious where the assistant should be placed. Was it part of the robot or outside the robot in the control-object? The most active participants wanted to place it inside the robot, since it was the robot's assistant. Some of the less active children were not so sure, and wanted a discussion. One of the active pupils says that the control-object sends the assistant to where it is needed. So when the assistant is needed in the robot, the control-object sends the assistant over to the robot-object, and then it can do what it is supposed to do. This is a very strange way of seeing how the program works. But it is understandable that the assistant is confusing, since it is not a part of the robot, but is still able to affect the robot.

The different sensors and motors had to be connected to the same ports on the RCX as in the program. This was something the different groups understood very fast and had little problems with. When changing from one type of sensors to the next, most groups deleted all their declarations and initializations of the particular sensor type and started writing it all over again. Because of this behavior I think they understand the correlation between the physical sensor and the sensor-object in their Java program. That they represent the same thing.

There was some discussion among the groups if they had to place their light-sensors on the same ports as the touch sensors had been placed just before. This discussion took place while writing the computer program after the the robot was built and the physical sensors were exchanged on the physical Lego robot. After some discussion they agreed that they could be placed on the same ports. I did not find this discussion in any of the other groups, so i do not think it was a common problem. There was no problem exchanging one of the sensors for another sensor on the physical robot, so why would it be a problem in the computer program?

5.10 Over-generalization

During an interview a boy wanted to have an assistant object in his object oriented house that was drawn on the piece of paper. This in an indication that this boy, and probably a lot of his classmates, used copy and paste a lot during their programming, and did not understand what they were copy-pasting. The reason he wanted to have the assistant in his house, was to get access to the wait method they had used when programming the Lego robots. This example of the misuse of the wait-method is an example of over-generalization (Boulay 1986). If you see bracket used some places in a code and not really understand why and how the bracket is used, you may use it wrongfully somewhere else. Very often in the children's programs they use the *assistant.wait* method both where it is necessary and where it is not. And in some cases it is used between each line of ordinary code.

The reason for wanting an assistant is to get access to the wait method here as well. This boy thought that if we did not have any means of slowing the program down, the house would create sofas at an alarming rate all the time. And this is not entirely wrong. It all depends on what kind of program we are talking about. If it was a house-simulator in the same way as we had a robot-simulator, we would need some way of slowing the program down. It is very understandable that they thought of the house-program as a simulator.

Another interesting observation was that some of the children had problems placing three objects of the same type in the house. Two doors were okey, but the third one felt very strange. If the third door was of a different kind, it was okey. This could be a result of the fact that they never placed more than 2 motors or 2 sensors of the same type on their Lego Robots. And therefore it was not possible to do that in a program. But it was not hard to convince them that was not the case. This is also an example that the children tended to over-generalize.

5.11 Using the Tools

All the children had a hard time remembering to turn on their robots before transferring the program they had written in Robolab, but the error message was quite clear. The children did not have much problems turning on their robots and trying to transfer the program one more time. A bigger problem was that the transfer bar on screen was a little too fast. It said that the program was transferred before it actually was, and this led to the children removing the robot before the transfer was complete. Then they had to start the transfer all over again. This is one of the five difficulties that du Bolay talked about in his article about the difficulties in learning to program: (Boulay 1986, p. 144). The difficulty of The Notional Machine. The children do not see the difference between the graphical bar on the computer screen, and the actual program transfer from the workstation to the RCX. They have no way to comprehend what is really happening inside the computer.

Another example of this kind of error was the fact that some of the children had a problem understanding what they actually did when they transferred the program. They would create a program on their computer, transfer it, then make some changes, and then test the robot again before transferring the new program to the computer.

5.11.1 Findings using Pilot

Making the robot go forward, turn and then go backward was pretty easily completed by most of the groups. But the Pilot version of Robolab also provided the possibility to use light and touch sensors. They were much harder to use than the timing icons. It was much easier just saying to Robolab that you wanted a motor to go forward for a given time, than telling the the robot to go forward until a touch sensor was activated. It was also difficult to build a physical robot that is able to crash into a wall and activate the touch sensor. We did not really want the children to use the sensors just yet, but since they were an option in the higher versions of Robolab, the children would naturally try them out.

This is a very typical example of children's curiosity and the value of bricolage. (Ben-Ari 1999) says in his article that bricolage is a way of learning by trying and failing, something that is usually a good way to learn when the subjects use a graphical programming tool. Since the children only had to choose the different icons forming the program, it was very easy for them to just test out the different icons and see what they did. And if the icon did not make the robot do what the children suspected, they would have very little chance of using the icon correctly. This was because the light sensor and touch sensor icons was not self explanatory in the same manner as the "motor go forward"-icon and the "stop after X number of seconds"-icon.

Most of the groups quickly advanced through the different levels of complexity from Pilot 1 to the most complex version of the program, Pilot 4. In the last level of the program you could have as many icons after each other as you wanted, seeing six at a time. Some groups made huge incomprehensible programs that just did different stuff, without really doing anything. If we saw this, we had their program deleted and asked them to start over again creating a program that made the robot do something smart. This way of treatment was well accepted.

5.11.2 Findings using Inventor

Our main goal of the experiment was to see how well the children were doing when they programmed the Lego robots using the Inventor part of Robolab. To be able to create a working program in Inventor you have to understand some of the programming and not just try to put different icons together and hope for the best. This is still a minimalistic way of teaching programming (Ben-Ari 1999). We only gave the children some basic training in using Robolab Inventor. The rest they should be able to figure out by themselves.

At first the Inventor was very big and incomprehensible. A good feature of Robolab was the different levels. When we used the lowest level (Level 1) of the Inventor the icons were pretty easy to understand. A tutor demonstrated how to place the icons on the desktop and how to sow them together. Then they managed to construct the same kind of simple programs they did using the Pilot version.

The children started creating their project assignment, first building the Lego robot, and then programming it. And they had mostly the same problems that they had using Pilot. It was not difficult to make the robot go forward, backward and turning, but using the different sensors caused a much bigger problem.

When a program was to be transferred to the RCX you pressed a white arrow on the top left of the Inventor windows. If this arrow was broken, the program had some kind of syntactical error and pressing the arrow at this state caused the program to show a new windows box with the error message, and where in the program the error was located. But this function was not very good, and the children had problems locating and fixing the problems. This was mainly because the children had a tendency to create huge programs without making sure that the different parts of the program worked. This also made the bug-tracking much more difficult.

When trying to reconstruct our demonstration robot the problem was to tell the computer program when the sensor saw a black line. We suggested that they should guess what integer represented the switch from tape to floor, but this was a very bad suggestion. We found out that we could get the robot to tell us what brightness it saw, and using this knowledge the children found out what level of brightness they were looking for.

The next problem was to find out what to do when one of the sensors saw a black line. This was something most of the children easily found out by themselves, since they had already learned to make the robot turn at will. So it was just a matter of deciding how long the robot should turn before it resumed driving forward. At the end of the last day most of the groups had managed to build a vehicle that could do something when it saw a black line.

Problems with Inventor

The biggest problem with Inventor was the sowing. To sow you had to combine two icons. But you had to combine them in the right way. The icons had 4 corners. One in, one out, and one for values and one last for other things. So when you wanted to sow 2 icons together you had to make sure you connected the right corners with the sowing thread. If you do not the program will not work, and it will be very difficult to see why it does not work. The sowing problem was the problem that caused the most confusion. This is also the biggest reason we think that Inventor is not such a good introductory programming tool.

One other big flaw in Robolab Inventor was that there was no way you could comment out a piece of code, and this was very frustrating because the children had a tendency to make giant programs that they themselves did not have control over, and when these programs contained errors, they did not know where to begin fixing them. So we, the tutors often deleted over half the students code and told them to start over again, programming some and then testing it before they moved on. This was not appreciated but they did as they were told. And this way of programming was much more successful. If Robolab Inventor had supported some way of structuring and splitting up their program into different parts, it would have been much easier for both the children and us tutors.

A smaller issue was that you had different kinds of mouse pointers. In our

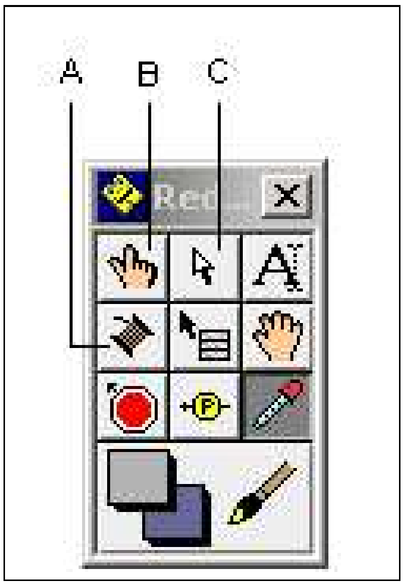


Figure 5.11: The toolbox in Inventor with the different mouse pointers

assignments the children had to use three different kind of mouse pointers. Figure 5.11 shows the different pointers. One pointer that looked like a reel and was used for sowing the icons together (A). One pointer for getting icons from the toolbox and moving the icons around on the desktop (C). And one last pointer for changing the value of an integer box (B). The toolbox with the different pointers were not on the desktop by default. And this caused a problem each time you had to restart Robolab. You had to press the window-bar on top, and tell Inventor to show the toolbox. We do not understand why the toolbox was not a default part of the desktop, considering its usefulness.

5.11.3 Findings using KarelJ and BlueJ

At first they told the KarelJ robot what to do step by step. Then we told them how to write and use while-loops and if-sentences. With just a little syntax-help the children were able to program pretty flexible robots that were able to pick up all the beepers in all the mazes presented. We even tried programming a messing-robot that put out beepers randomly in the world, and they had to make a flexible cleaning-robot that had to clean up the mess. And some of the children managed to write quite complex programs picking up the mess. But the messing-robot had a bug which made it impossible to create a 100% effective and flexible cleaning-robot.

This way of programming was very different from the way the children were used to programming robots. There are very few similarities between Robolab and KarelJ-BlueJ. Still the children were able to construct pretty flexible and complex robots. There could be two reasons for this. Either the programming tasks the children were given were too easy and they only had to apply some copy-paste technique in order to complete the programming tasks they were given. The other solution is that they learned a lot of robot-thinking using the physical Lego-robot first. And they were able to use the same kind of thinking when programming the KarelJ-robots. It is probably a good idea too keep the robot-domain when switching to a new programming-domain. When doing just that the children will have fewer new things to learn and can concentrate on the programming.

5.11.4 The API and the help web-page at Mølladammen

Even though the participants got a handout of the API for our version of Lejos, they did not use it much. One reason for this was that it was not easy enough to use, but the main reason was because they do not have any training at all using any kind of manual. We told the children to use the API to solve a problem, or to find out i.e. how to make the Robot play a sound. In most of the cases when we did this the children managed to solve the exercise without much help from us. But usually when the children had a problem they just put their hand in the air and called for help, without trying to find the error themselves. As mentioned in chapter 5.2 on page 60 I think this was because it was an easier and less time consuming way of getting to the goal of finishing the assignment.

When working with the restaurant assignment, the groups were much better at using the API that was delivered to them, than they were in the previous assignment. When programming the Lego robots the assignments were described on the help web-page and they did not receive a printout of these assignments. During the restaurant assignment, both the assignment and the API was on paper. In a lot of the groups one person would be reading the API and advising and helping the one at the keyboard. And this seemed like a successful way of working together. An example of pair programming where one sits at the keyboard and is the driver, and the other sits beside him and acts as a navigator.

5.11.5 Using Jcreator and RCXDownload

Jcreator was the editor we used during the java writing assignments, both at Fjellhamar and Mølladammen. This editor was easy to use, and none of the children seemed to have any problems using the editor to write or copy and paste their Java code.

RCXDownload was used to compile and transfer the code written in Jcreator. The children easily managed to select the Java file they wanted to compile, and if no errors occurred during, they just pressed the transfer button, and the program was transferred to the RCX.

When using one program to make the code, and another to compile it and send it to the robot, it is easy to forget to save code before compiling it. Even expert programmers have this problem. And it caused a lot of confusion both among the teachers and the children. But in most of the cases the children were good at finding out what the problem was. This was probably because every time they wrote something new, the response from the robot should be somewhat different from last time. And it was also easy to see that the robot did what you wanted it to or not.

5.11.6 Hardware problems

There was some problems related to the use of the Lego sensors. The most difficult sensor to use was the light sensor. One of the assignments was to make the robot follow a black tape-line on the floor. The children would often start programming right away and did not bother to find out how the light-sensor really worked. They needed it to tell the RCX when it saw something dark enough to be the black tape and not just the floor. This was a common problem with most the groups. Some of the groups started with a small test program using only one sensor, and just making the robot stop when it hit the sensor. These groups got a lot longer with their programming than the ones programming it all right away. A problem with understanding the light-sensors, was that even though you got no reaction when the robot was on the floor, it reacted as soon as you lifted it up. It was very frustrating, for the children, to see the robot reacting when it was not supposed to.

The next thing to do was finding the correct light-value that showed when the light-sensor saw a black tape instead of the floor. The children guessed and used a trial and error-method. Some groups were told that they could print the light-value on the display of the robot, and were shown how to do this. In that way they could find out what value the floor was and what value the black tape was. But even when this was taught to the children, they had problems understanding that the middle-value between these values were a good value to look for. I guess this requires more general programming knowledge to figure out. There were also a problem that the difference between the tape-value and the floor-value was so small, usually under ten, that if they placed the sensor somewhere else later on, they had to reconfigure the light-values.

The amount of light the Lego light-sensors saw varied a lot from robot to robot. The reason for this was the distance the sensor was placed from the floor, and how sunny it was outside. If the sun suddenly moved behind a cloud, some of the sensor values had to be reconfigured. This took a lot of time, but was unavoidable. Another problem was getting the vehicle to run slow enough for the sensors to see the black lines on the floor. This was not a big problem for them who had their robot with belts instead of wheels.

When trying to make the robot turn left or right different strategies are tried out. This was a task most of the children managed without much difficulties. When the Java API was used, the starting template had two motors, and both of these motors were told to go forward at the same time, thus making the robot run forward without turning. And using this template the children experimented. One strategy was to tell the different motors to go forward at different powers. They could choose between zero and seven, where seven was the strongest and the fastest. So when they told one to go at a speed of four and the other at a speed of seven, they thought the robot would turn pretty smooth. And this was something they wanted the robot to do when following the black line. The outcome of this strategy was usually that the robot failed to turn enough. This was even a problem when they used power level two on one of the motors, and power level seven on the other motor. This was a bigger problem with the wheeled robots than the belted ones. And the reason, of course was the friction between the rubber tires and the floor was too great, so the most powerful motor would cause the whole robot just to go forward. Another way of making the robot turn was to stop one of the motors completely. And this worked a lot better with the belted vehicles. With the wheeled ones, the forward tires would still keep the robot from turning. Some of these groups removed the rubber from their forward tires, and that helped a lot. But this was not something they figured out by themselves.

I guess they had not learned much about the physics of the real world. This is one of the downsides of using real world physical objects. A virtual robot would not be subject to physical laws, like friction. During the experiment we had a lot of problems because we used this kind of control-technology. Sometimes the children would make a symmetrical robot and tell both the motors to go at the same power, and the robot would still turn slightly.

A Lego Robot is not very sturdy, and the different constructions that the children had made kept falling apart. We had decreased the difficulties of building a sturdy robot a bit, since we had created easy to use building-instructions. The batteries had to be exchanged all the time, and every time they were exchanged, we had to upload the lejos software to the RCX, and this took five minutes to do. Transferring the compiled program from the workstation to the RCX was also very unreliable. It took over a minute to transfer a small program, and since the RCX relied on infra red beams to communicate with a workstation the transfer was often interrupted. And the transfer had to start all over again.

One of the groups had a strange idea of what the Lego-robot consisted of. When asked to describe what their robot consisted of, they said that the motors leftMotor and rightMotor had one sensor each. I find this a strange thing to say. They may think of the RCX as a motor, and when they say that it has different motors they may be thinking of the different ports of the RCX. This group was very insecure. None on the group dared to say much

A girl on one group said this about the difference between the sensor and the motor: “The motor is the motive power, and the sensor have to use the motive power to do something” I am not sure what she meant by this, and a little while later she said that she did not know what she was talking about. It could have something to do with having to use some kind of power in order to do something. Or it could be a matter of input/output thinking, since the motor has some kind of output(the wheels that go round), and the sensors only being able to sense things as an input-device. But this is just speculations. Somehow she thought that the motor created the energy the sensors used. The other people on her group convinced her that this was not the case. The RCX works as a big battery, powering the motors and sensors.

During the experiment there was no one designated as cameraman that had the responsibility to point the camera to the most interesting group and switching to the correct computer screen. This was done by a random tutor that had the extra time. During the project there were between two and four tutors, and this was not enough to have one of those tutors constantly keeping track of the cameras. This resulted in a quality loss in the material captured by the cameras. Another problem was that the streaming server had problems encoding the larger movie-sessions. When the camera had been filming continuous for two hours and we wanted the streaming server to finish the session, the streaming server stopped working and the whole session was lost. So we found out that we better have shorter sessions.

I estimate that we lost approximately a days worth of filming material because of this problem. The strange thing was that this problem did not occur when we used the same material to film the children at Fjellhamar one month earlier.

5.12 Problems with the assistant

The assistant was never really a part of the robot or the robot-program, it was just an annoyance that we had to learn the children to cope with. It created a lot of unnecessary confusion when the children were asked to draw their robot. It was difficult to determine where the assistant really belonged, if anywhere. Even worse was it if the children wanted to make a method in the robot class, i.e. turn(). They had to find out that they could not call the motor using the same syntax they used calling the motor from the control-class. And at the same time they had to make a new assistant-object in the robot-class in order to get hold of the wait-method. This created a lot of unnecessary confusion. Figure 5.12 on page 91 shows how the assistant has to be declared both in the control class and in the robot class. If we could, we would have dropped the assistant. But we needed the wait method. We tried to explain that the assistant sat inside the vehicle and controlled it, but this analogy was completely wrong when we said that it was not really a part of the robot.

During an interview one of the groups were asked what objects they placed on their Lego robot in the Java program they wrote. Among the objects mentioned was the assistant object. This was not supposed to be a part of the Lego robot object.

When asked again if they really placed the assistant on their robot, the answer was no. The place for the assistant is in the control-object. Another group said that the one writing on the LCD-display was the assistant. This was not the case, since the display-object was made in the robot, and did not have anything to do with the assistant. A reason for this LCD misunderstanding could have been that the children did not physically place the LCD-display on the Lego robot. It was just there from the beginning. And this can be compared with the assistant task of getting the program to sleep.

```
class Control {
Robot volvo;
Assistant per;
        Control(){
        volvo = new Robot();
        per = new Assistant();
    per.wait(4000);
    volvo.leftMotor.forward(5);
    volvo.rightMotor.forward(5);
    volvo.turn();
}

class Robot{
Legomotor leftMotor;
Legomotor rightMotor;
Assistant nils;

Robot(){
        leftMotor = new Legomotor(A)
        rightMotor = new Legomotor(C)
        nils = new Assistent();
        }

        public void turn() {
                rightMotor.forward(3);
                leftMotor.backward(3);
                nils.wait(1500);
        }
}
```

Figure 5.12: A piece of code that shows how the assistant has to be made in the robot-class in order to get the turn-method to work.

Chapter 6

Summary and Conclusion

6.1 Children and Lego Mindstorms

Lego bricks are something that almost all norwegian children have played with at one time or another. The building bricks are very flexible and easy to use. I dare to say that every child over 7 years in age is able to build a small house out of Lego bricks. This makes Lego a good choice when using it in a learning context. The problem with Lego Mindstorms is that it is basically based on Lego Technics, and this is a kind of Lego that fewer of the children have used and are familiar with. The children showed little knowledge of the use of cogs and shafts, and during the experiments a lot of time was used building sturdy functional robots.

Both the 11 old year boys and girls found this way of working with control technology interesting and fun. So did the 14 year old boys. But the 14 year old girls did not have the same enthusiasm for Lego Mindstorms.

This use of control technology is more interesting for boys than it is for girls at the age of 14. The results from these experiments show that boys have an easier time relating to robots and computer programming. The boys were much more interested in programming the robots they had built. The girls became bored much faster and started to wander from their group and disturbing the other groups. But girls had no harder time understanding how to use the control technology than the boys had.

It was also obvious that the boys are "supposed" to know more about technical Lego than the girls. Chapter 5.2.2 on page 61 shows that when asked how the physical robot works, the girls hesitated to give an answer, and wanted the boy to answer instead. Even though the girls probably knew as much as the boy did.

6.1.1 Working together

The children worked together in groups of three people both at Fjellhamar and at Mølladammen. At Fjellhamar there were single sex groups, while at

Mølladammen the genders were mixed, either with two girls and one boy or two boys and one girl. At Fjellhamar three people on the same group was not such a bad choice as it was at Mølladammen. At Fjellhamar one of the group members could build on the Lego robot while the others were programming, or two could build while one was programming.

At Mølladammen two people on one group worked together and programmed in Java, while the third person wandered off to disturb another group. Or one person worked with the assignment while the two other people sat in the background and talked about something else beside the assignment. In the last case, the person working alone with the task often just had to turn his head and ask the others a question, and they would reply with a correct answer and help the worker. This leads me to believe that some of the people that did not show a big interest in the experiment did not find it challenging enough, and lost their motivation. But they were able to help schoolmates if they were asked to do so.

Still I feel that two people on each group would be better than three. Because we just had a limited number of Lego Mindstorms sets and nine groups are already too many groups, because since we only filmed four groups, the other groups just took too much time when they needed help. The third person on each group wandered a lot around and disturbed the other groups. The only exception was the group with three girls. They cooperated better than the other groups, and none were left behind. This might have been a coincidence, but maybe children at age 14 work better together if they are of the same gender. Since the 11 year olds were better at working together, and they had single sex groups, I conclude that at the age of 14 and below you work better with people of the same sex.

6.2 Object oriented concepts with Lego Mindstorms and Lego

The programming environment from Lego called Robolab could not be used as a tool for teaching object oriented concepts. There was no possibility to use encapsulation. In addition the Pilot version was too simple and not flexible enough, and the Inventor had too many design bugs, which made it unnecessarily difficult to use.

Java and Lejos together with Lego Mindstorms make up a better teaching environment for object oriented concepts. Even though Java is difficult to use for beginners, the children were able to learn a lot of both programming and program design during the experiment.

According to my results I would claim that the children understood part of the program structure of the object oriented Java program. They understood that an object needed a name in order to use the object in the computer program, but I do not think they understood that the name of the object was a pointer to the object and not the actual object itself. The results also show that they understood that they had to make an object representation in the Java program in order to make the physical equivalent object move. I.e., in order to be able to type something on the display of the RCX they had to make a display in their Java program.

The results from the experiments, especially from the interviews at Mølladammen, show that the children understood a lot about the program structure of their Lego robot Java program. They were very good at drawing an object model of their robot on the paper that was handed out, and looking at the drawing they were very good at writing the method calls I wanted them to write. This was done while looking at their drawing. They managed to follow the lines from the control-object to the object that contained the method they wanted to use. And when changing domain and drawing an object oriented house instead, the children performed just as well when writing the method calls I asked them to. It seems that they learned something from programming the Lego robots that they could use in a more general way. I draw the conclusion that some of the children understood the connection between the dot-notation they used while programming the Lego robots, and the connection between the objects in the computer world.

Because of the way you declare and assign objects in Java, the children never understood why you had to type:
LegoMotor leftMotor;
at the top of the class and
leftMotor = new LegoMotor(A);
a little further down in the same class. During the programming they kept forgetting either the first or the second.

It was easy to understand how to use a number or a letter in a parameter after assigning a new motor or sensor like in the example above. But using an object in a parameter was much more difficult. When asked, the children explained their reason for typing
volvo.leftMotor.forward(5);
It was because *"it is the volvo's leftMotor that is ordered to go forward"*
But during the interviews and the restaurant assignment when they needed to use an object in the parameter, they failed to tell Java where the object was located. As the discussion in 5.9 on page 76 shows, the children had problems understanding that when they were supposed to do something to table4 in the restaurant, they had to type *objecta.table4* and not just *table4*. Even though they answered that table4 belonged to the restaurant objecta, they failed to type this in their program code. My conclusion to this is that they did not really understand how the program structure was. They managed to get it right when using a method, but this was probably because they copied and pasted it from somewhere else in their program code.

A good way to learn the principle of encapsulation would be to make methods in the robot object. During the experiment only one group made such a method in their robot, a method called turn. The main reason almost no groups made this kind of method was that we did not explain to the children how to create such methods. The only group that made one was helped a lot by a tutor. We hoped that the students would see the similarities between the pre made classes like LegoMotor and ask the tutor how they could make methods themselves like the one in the LegoMotor objects they made. During the experiment no such questions were asked, so I feel that we should have used some time explaining about methods before starting to program the Lego robot.

Another reason for the absence of such methods was that the assistant was originally placed in the control class only, so in order to access the wait method you had to create one assistant in the robot class as well. And this caused a lot of confusion. But as mentioned in chapter 5.12 on page 89 we should have found another way to make the Java threads sleep.

All in all I think the children understood what object was supposed to do something when they wrote calls like:
"volvo.legoMotor.forward(6);"
But I do not think they understood how the parameter worked. The problem when you needed to place an object in a parameter, shows that this was difficult to understand. But I guess the understanding would have been greater if more groups had made their own methods in the robot class, that they could call from the control class.

6.3 Lego robot programming in Java and the real world analogy

It seems from the results presented in chapter 5.9 on page 78 that the children had an easier time understanding how to use the methods from the restaurant API when they had a connection with the real world. When the waiter would take the orders, he would talk to the guests one at a time and writing down their order. And this was something the children managed to do with much ease while programming the restaurant. Seating the guests one at a time instead of seating them all at the same time was more confusing. They would rather type:
"objecta.waiter1.seatGuests(guest1, guest2, guest3, guest4, objecta.table4)"
instead of typing
"objecta.waiter1.seatGuest(guestX, objecta.table4)"
four times. So if we had managed to make the restaurant more like the real world the children would probably have had less problems using the restaurant methods correctly.

When building an object oriented house on a piece of paper, the children were able to create rooms inside the house and furniture inside the rooms. But when the object that was going to belong to the house, reprented something huge the children had difficulties understanding how it could be a part of the house. An example is when the children were asked to make a garden a part of the house. This caused a lot of confusion.

This shows that as long as the program structure correlates to the real world, the children understand how to create an object oriented program. And they understand how to use an object oriented program. But when this correlation is missing, the children are confused and have a harder time understanding how the program works.

The children had a tendency to assume that the computer knew what they wanted the computer to do. It was difficult to understand that the computer was stupid and had to be spoon-fed. (Pea 1986) talks about the "superbug". The idea of a hidden mind inside the computer that understands what the programmer wants to achieve. And some of the errors performed by the children can be explained by this incorrect assumption.

6.4 Using the tools

6.4.1 Using Lego tools

When working with physical objects, like Lego Mindstorms robots, a lot of things can go wrong. Building a sturdy robot with the bricks provided is a difficult task that takes a lot of time. When working with a computer program on a computer screen, like KarelJ, you do not have to worry about physical problems. See (Borge 2004) for an example of an experiment where virtual robots were used instead of physical robots.

Using the light sensor was very difficult. When the sun outside moved behind clouds and out again, the light values used for detecting a black tape on the floor also changed. So the children needed to re-calibrate their light sensor values all the time, and this was unnecessary frustrating and time consuming.

During the experiment at Fjellhamar we found that we were not able to find any ways of teaching object oriented concepts using the programming language from Lego: Robolab. In general programming we found that the Pilot version had too little flexibility and the Inventor version had a huge flaw. This flaw was the way you sowed the different icons together. (See chapter 5.11.2 on page 84 for a more detailed description of this problem). The children were still able to create complex problems that used both loops and if-statements, but this was something they managed to do using KarelJ and BlueJ as well.

6.4.2 Using custom made tools

The text editor used when writing Java code was Jcreator. Together with the RCXDownload tool for compiling the Lejos Java code and transferring it to the RCX this was a combination that worked very well and was easy to use. The children were good at saving their program when they had made changes, and then using RCXDownload to compile and transfer their program. The error messages provided by the compiler were normal Java error messages. Understandably these were difficult to understand by the subjects, but I do not think that it made any difference that these error messages were given by another program than the editor.

6.4.3 Documentation - paper vs. digital assignment description

The children were not very good at using the Lego help page we provided for them. The printout of the API were more often used than the pieces of code provided by the web-page. They were also much better at finding a solution to their problems using the API, rather than reaching a hand in the air, when programming the restaurant, compared to when they were programming their robots. This was probably because the assignment tasks themselves were on a piece of paper, and not on the help page, when the children were programming the restaurant.

6.4.4 Tools summary

All in all I would say that using physical control technology, like the Lego robots, has a lot of time consuming drawbacks. But for 11 year olds I think this way of working is much more entertaining than using computer simulated robot program. The 14 year olds were tired of playing with Lego after two days. The last day they used a computer simulated program, namely the restaurant simulation. They old welcomed the change.

6.5 Assistant

To be able to make the Lego robot go forward for a given amount of time, we needed to use Java's sleep method. But we wanted this sleep function to be part of the package we had prepared for the children. The sleep method is part of the static Thread object. We did not want the children to have to relate to more than necessary, so we wanted to hide this fact, and rather just use the sleep-method like any other method the robot could use.

The solution to this problem was to create an assistant that had a wait method. But as you can see in this thesis, placing this assistant in the control class created a lot of unnecessary confusion. Another way to "hide" the sleep-complexity would have been to put the assistant in the robot class. Then when the children wanted the robot to go forward for a given amount of time, they would use the wait method in the same way they used i.e. the forward method:

volvo.rightMotor.forward();
volvo.per.wait(1000);
In the example above "volvo" is the name of the robot and "per" is the name of the assistant. A third option would have been to make the wait method a part of the robot class or the control class. But we wanted the wait method to belong somewhere, like all the other methods did.

I think now, that the best solution would have been to place the assistant object in the robot class. Then it would seem that the robot waited until it did something else, and that is an analogy I think would have been easy to understand.

6.6 Hints for similar experiments in the future

It would be wise to spend a day or two just playing with the Lego and getting used to the camera, before actually starting to gather result data. Very much time was spent in vain looking through videotapes of children playing with Lego in a less constructive way. And since the experiment only lasted for three plus one day, we would have gotten more interesting results if more time was spent programming instead of fooling around. This is very logical, that the more time you use gathering data, the more relevant data you will get. But if i.e. the school had used a couple of days filming the people we would be filming, and just playing with the Lego as a spare time activity. And me and the other tutors would not have to be a part of it. And then when we came and did what we wanted to do, the children would have gotten used to the cameras and the Lego. But I guess this is just using the schools valuable time so that me and the tutors would not have you use so much of our own valuable time.

I would also recommend having one person dedicated to follow the most interesting group with the camera. This person will see to it that when a group of participants work on a problem, the whole session from finding the problem to working out a solution would be recorded. This was something I really missed when going through the recorded material from the experiments described within this thesis. This requires that one more person has the time to participate in the experiment, but I think it will be worth it.

At last I would like to mention that when you decide to have a pilot experiment before the "real" one, you should use more time looking through the results from the pilot experiment before the second experiment takes place. This was something we failed to do, and we would probably have been better prepared for the second experiment if we had more time between the experiments.

6.7 Implications for teaching

Teachers that plan to use Lego Mindstorms and Java as tools for teaching programming and object oriented concepts should have very well defined assignments for the children to solve. And the assignments should have an increasing difficulty, so that the children are challenged all the time.

Children between 11 and 15 are also very interested in competing with each other. They put a lot more effort into what their doing if it means that they can beat someone else. This was very obvious when looking at the children at Mølladammen. They wanted to solve all the assignments as quick as possible in order to be better than their schoolmates.

It is important that the assignments are created in such a way that taking shortcuts is difficult. The children do not always want to learn, but they like solving assignments. What I mean is that their main motivation is getting the correct result from the assignment and not learning.

Before introducing Java and the Lego it would be wise to introduce the object oriented concepts that you want to teach. You could make the children draw a house, or something else, on a piece of paper. Doing this in the same manner as I did during the interviews at Mølladammen can be a good way to introduce encapsulation and inheritance. Make them i.e. draw the house they live in, the different rooms, the furniture, and the family members. And let them give the different objects in the house different properties and abilities. This may be very boring for the children, at least if they are teenagers or older, so you should only spend some hours on it.

A lot of time was spent building the Lego robots so that they would be sturdy enough. This time can be saved if building templates are handed out, and only the pieces needed to build a simple robot are given to the children. Then they can concentrate on the programming and they do not have to be frustrated about a Lego robot that keeps falling apart.

Use some time before the building starts to explain how the different sensors works. Especially the light sensor. This sensor is difficult to use, and it is difficult to understand why it is not working the way you want it to work.

Put the children together in pairs. Groups of two work much better than groups of three. You should probably use single sex groups as well. It will lead to more competition between the groups, and it will make the assignments more fun. And there will be less competition inside the group, so the members of the groups are more likely to tell each other what they think they know, instead of holding it to themselves and never dare to ask the group mate.

And if you are able, you should get the children to explain to each other what they are thinking and discuss among themselves. (Holmboe 2004) recommends the same in his article. This will give both the tutors and the children a deeper understanding of what they are trying to find out.

A couple of times a day there should be a gathering in another room without the computers and the Lego. When more groups find something difficult, this should be discussed in these groups, and children who think they have an answer should be allowed to explain to the other students what he or she thinks. This way the children will learn from each other, which is a much better way to learn something, than from a teacher in front of a blackboard.

These project days should never be more than two days in a row. Children get tired when working with difficult assignments. But if this course could be a course like any other course, i.e. two double classes during a week, the children would have had something to look forward to, and maybe think about the things they have learned.

6.8 Comparing my results with the results from Borge(2004)

(Borge 2004) wanted in his thesis, to find out if the use of graphical environments is good tools for introducing object oriented programming the objects first way. His conclusions indicate that the use of graphical environments help the students visualize how their code affected the movement of the Java objects. This is something I also feel that the use of physical Lego robots did for the children in my experiments. When they made a little change in their code, the robot did something different, and the children understood what in their code, made this difference.

Borge complains that he had too few participants to be able to draw any strong conclusions. My experiments had too many participants, but not enough recording tools to manage all of them. So we concentrated on four groups in each of the experiments.

Both the experiments described in this thesis, and the ones in Borge's thesis concludes that the subjects learned a lot of the object oriented concepts, got an understanding for object oriented programming and how to make their own objects before learning more imperative programming. In all the experiments, little time was used teaching procedural programming, like loops and if-statements. They also tried to avoid introducing the main method and the use of the Java phrase static.

Both theses conclude that it was fun to use graphical/physical environments to learn and program. But none of the environments are used much in teaching and need further research. They need further study. Study by experts before being used more in experiments. But we agree that this might be the way go.

Borge talks about using BlueJ and KarelJ together. This was something we did, but it is not well described in this thesis. But at least we got the environments to work together. And the children at Fjellhamar managed to solve our assignments, so it could not have been all that bad.

Bibliography

Atheron, J. S. (2003), Learning and teaching: Assimilation and accommodation. http://www.dmu.ac.uk/ jamesa/learning/assimacc.htm.

Ben-Ari, M. (1999), 'Bricolage forever!', *Proceedings of the 11th annual workshop of the Psychology of Programming Interest Group* .

Berard, E. V. (2000), Abstraction, encapsulation, and information hiding. The Object Agency, http://www.itmweb.com/essay550.htm.

Berge, O. & Fjuk, A. (2003), Socio-cultural perspectives on object-oriented learning. Presented at ECOOP 2003.

Bergin, J. (2000), Why procedural is the wrong first paradigm if oop is the goal. http://csis.pace.edu/ bergin/papers/Whynotproceduralfirst.html.

Borge, R. E. (2004), Teaching oop using graphical programming environments, Master's thesis, University of Oslo.

Borge, R. E. & Kaasbøll, J. (2003), What is "oo first"? Presented at ECOOP 2003.

Boulay, B. D. (1986), 'Some difficulties of learning to program', *J. Educational Computing Research, Vol. 2(1)* .

COOL (2002).
*http://www.intermedia.uio.no/cool/

Denis, B. (1993), *Control Technology in Elementary Education*, Vol. 0, Springer-Verlag.

Detienne, F. (2002), *Software Design - Cognitive Aspects*, Springer.

Greeno, J. G. & Collins, A. M. (1996), 'Cognition and learning', *Handbook of Educational Psychology* .

Groven, A.-K., Hegna, H. & Smørdal, O. (2003), Oo learning, a modeling approach. Presented at ECOOP 2003.

Holmboe, C. (2004), The linguistics of object-oriented design: implications for teaching. Accepted for Educators Symposium, OOPSLA2004.

Kolikant, Y. B.-D. (2004), ‘Learning concurrency as an entry point to the community of computer science practitioners’, *Computers in Mathematics and Science Teaching* .

Lattanzi, M. & Henry, S. (1996), ‘Teaching the object-oriented paradigm and software reuse,’, *Computer Science Education, V7, N1* **7**(1).

Lego Mindstorms (n.d.).
*http://lejos.sourceforge.net

Lego Mindstorms for schools (n.d.).
*http://lejos.sourceforge.net

LEJOS. Java for the RCX (n.d.).
*http://lejos.sourceforge.net

Meyer, B. (1988), ‘Object-oriented software construction’, *International Series in Computer Science* .

Papert, S. (1980), *Mindstorms - Children, Computers, and Powerful Ideas*, BasicBooks.

Pea, R. D. (1986), ‘Language-independent conceptual "bugs" in novice programming’, *Journal of Educational Computer Research* .

Robins, A., Rountree, J. & Rountree, N. (2003), ‘Learning and teaching programming: A review and discussion’, *Computer Science Education* .

Shackelford, R. L. & Badre, A. N. (1993), ‘Why can’t smart students solve simple programming problems?’, *International Journal of Man-Machine Studies* .

Stroustrup, B. (1994), *The Design and Evolution of C++*, Addison Wesley.

Valcke, M. (1993), *Knowledge Representation and the Learning Process: Taking Account of Developmental Features and Support Features in Interactive Learning Environments*, Vol. 0 of Denis (1993), pp. 13–25.

Appendix:

Roar Granerud, Jens Kaasbøll, Richard Edvin Borge and Christian Holmboe

Department of Informatics, University of Oslo

{rgraner, jensj, richared, christh}@ifi.uio.no

Children's Understanding of Object-Orientation

Abstract

Previous studies have shown that adults can learn object-oriented programming through an objects-first approach. The experiment reported in this paper demonstrates the feasibility of objects-first also when teaching 14 year olds. The software tool used for teaching, Lejos, defines the classes to be used, and objects of these classes have physical counterparts in the Lego Robolab components that execute the children's programs. Lejos required programming with exceptions, which was hidden by a shell that was constructed for the experiment. A procedural, graphic programming package, Mindstorms, also to be used with Robolab, had flaws in interface design and mechanisms for structuring imperatives.

Introduction

A previous literature survey concluded that novices preferred a procedural structure of their analysis of a domain, which corresponds to imperative programming rather than the object-oriented approach (Détienne, 1997, p60). Recent studies of learning object-oriented programming have shown that novice adults can learn to create subclasses, create objects and call methods after one day of training, even though they could not distinguish between references and objects at that stage (Borge, Fjuk and Groven, 2005). A condition is that the domain is simple and that the objects created are visualized. A study of mid term mastery in a beginners' OO procedural first course, showed that the students were able to use the basic OO concepts and syntax, but they struggled with program design, and they tended to code directly without sketching models (Kaasbøll et al, 2004).

Previous complaints by professors who state that OO is too difficult to learn during the first year of study have thus been refuted. The research seems to show that adults are able to learn programming concepts in the sequence these are taught, and that they struggle with translating the problems into the formal expressions required and designing their programs, while the syntax and semantics of the programming language constitute smaller obstacles to their learning.

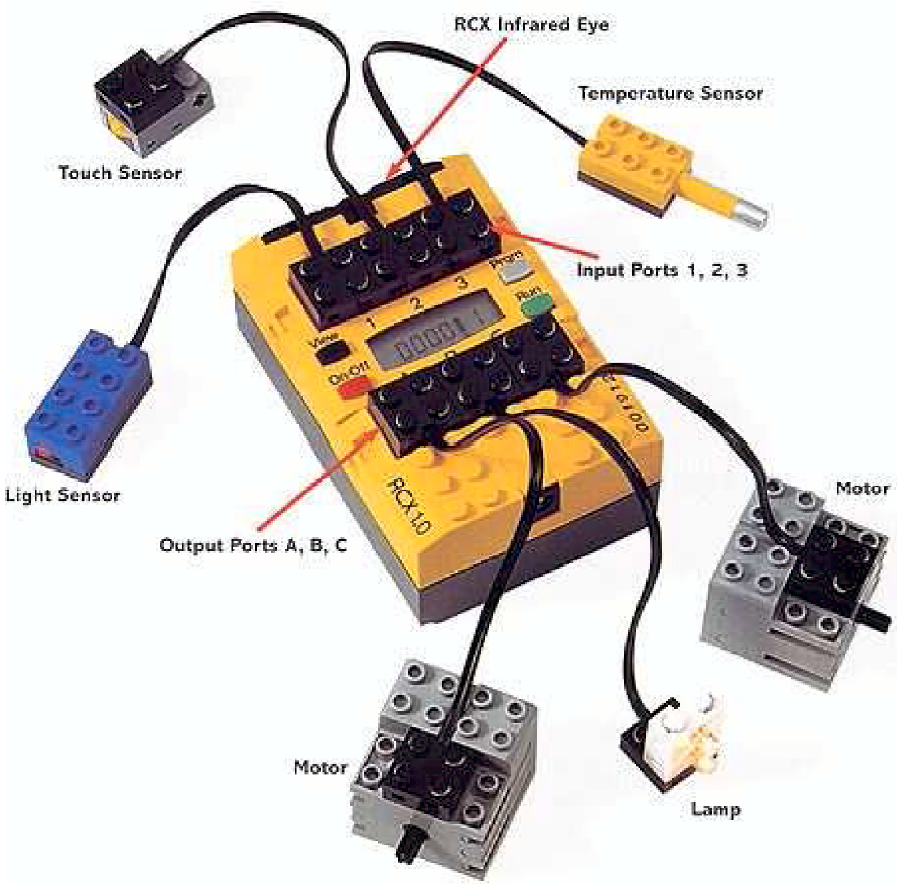
In order to know whether these findings are age specific, and whether teaching object-orientation can be done in schools, two experiments with teaching OO to grade 6 and 9 classes were carried out.

Tools

Robolab

The LOGO.language (Papert, 1980) is designed to program floor turtles that can be used to draw different geometrical figures on a piece of paper. Children have used LOGO for learning imperative programming. The idea has ben taken up by Lego (2005) in their design of the toy construction set Mindstorms and the Robolab (2005) programming environment.

The central part of Mindstorms is a control brick (RCX), containing three ports to motors or lamps, and three ports to sensors, a display, a loudspeaker, an IR eye for receiving programs from a PC, and batteries, see Figur 1. The sensors can detect light, touch, rotation and temperature. Robots like cars have to be built with motors, cog wheels and the control centre.



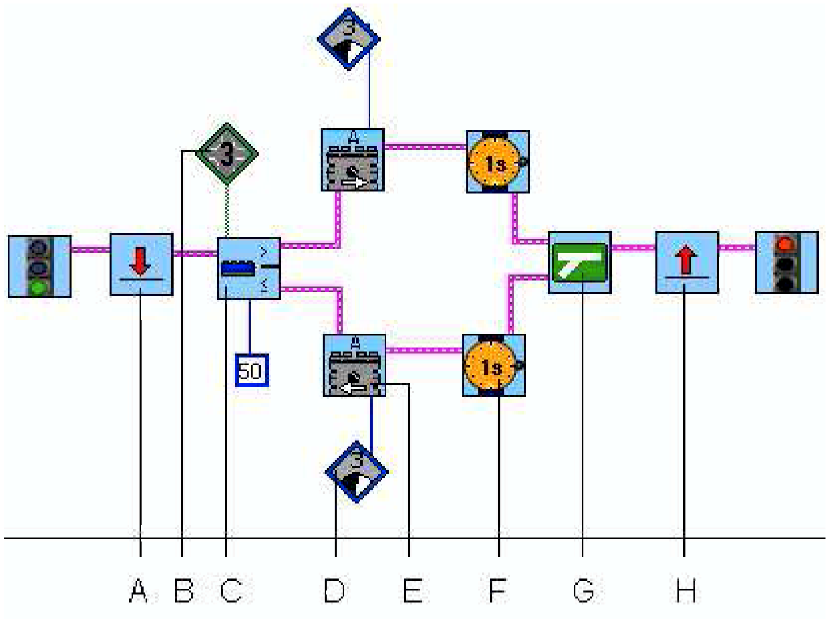
Figur 1. The RCX with motors and sensors attached.

The Robolab programming environment has two user levels: Pilot and Inventor.

Pilot is divided into four levels, each with increasing complexity and functionality. At the first level you are given a working program consisting of two icons, one to signal the motor to move forward, and one to say for how long. These icons can be exchanged with similar icons, i.e. making the motor go backward and changing the time before full stop.

At the Inventor level you start with an empty desktop except for the start and the stop icon, represented as traffic lights. To build your program you have a tool case with different icons representing the actions the robot can perform, and additional icons representing numerical value. You drag and drop the icons you need onto the desktop and move them around using the mouse. Some icons represent if-forks that read sensor values. The sequence of operation is determined by a thread that has to be sewn between the icons.

Figur 2 shows a small program using an if-fork. The program checks if the light-sensor value is higher or lower than 50 (C). If it is higher, the topmost route is chosen, and if it is lower or equal the other will be used. Icon B represents the port on which the physical sensor is mounted on the RCX. Icon E represents a motor mounted on port A on the RCX. It will go at a power of 3 (D) and will do so for 1 second (F). The difference between the routes is the direction of the motor. The paths are merged (G) and icon H makes it jump to the landing-icon (A).



Figur 2. A Robolab Inventor program with an if-fork (C) and a goto (from A to A).

Lejos

Lejos (2005) is a Java tool for programming Lego robots with classes Motor, Sensor, etc. Its API requires knowledge of Static, Main and Exceptions, and experienced programmers used a substantial time to learn utilizing it. In order to hide complexity, a shell was programmed so that all threads and exceptions were hidden through two classes called Control and Assistant, which had to be instantiated in the students' programs.

KarelJ with BlueJ

These tools are described elsewhere in this volume (Borge, Fjuk and Groven; Hegna and Groven). In this experiment, KarelJ was run through BlueJ, in order to have a visual interface also of the code.

Restaurant simulation environment

An environment was built in Java for simulating guests ordering dishes from waiters, chefs cooking and waiters delivering the dishes to the correct tables (Hegna, 2003). Class Guest with methods like chooseFromMenu, class Waiter with placeByTable(guest,table), receiveOrder(guest), findCorrectGuest(table), etc. The environment opens for generating objects and writing method calls closely resembling the business logic that the pupils have experienced when visiting restaurants. No visual representation of the objects appeared at the screen, when executing the programs, ASCII output like the following appeared:

```
Hanne has received Ice cream as a desert. And this is correct.
Per has received Salad as a main course. Incorrect. Per ordered Beef.
```

Method

The class of 11-12 year olds consisted of 27 kids in a primary school, and the 28 fourteeners were in the second of three grades in the mandatory, lower secondary school.

The 11 year olds were given three days of teaching, mainly with procedural programming of Lego Mindstorms with Robolab. One OO activity in between consisted of a half an hour paper and pencil exercise on specialisation. The last day the kids used KarelJ and BlueJ.

The 14 year olds were instructed by means of Java programming of Lego robots for two days, while the last day was devoted to Java programming in an environment with the restaurant simulation.

All sessions were videotaped. This was done by cameras that were able to film both a group of children and their computer screen at the same time. The discussions were also recorded with a microphone. The camera focussed on some of the children, and for periods, those who seemed to be of most research interest, were selected for recording. Out of 35 hours of teaching, around 25 hours were video recorded, partly due to a technical breakdown one day.

Programs and paperwork done by the children were collected.

During the last day of the experiment, four groups of pupils were interviewed, two of which were selected due to their mastery of the programming, and two more at random

The videos were viewed in order to take notice of the specifically OO issues that the children worked on. Also other issues of possible interest, like intense cooperation and clues of motivation were noticed. Relevant video intervals were then transcribed.

Studying with hardware and software

Age and gender

All of the 11 years old were eagerly using the tools for the three day period. Some boys spent most of their time constructing the mechanics of the cars and had little interest in programming the robots. Two groups consisting only of girls and one of boys were constructing complex programs. These groups, constituting 1/3 of the class, also picked up on imperative Java programming when this was presented during the last hour of the three days.

The 14 years expressed that Lego was for kids, and less enthusiasm was observed. The 14 year old girls became bored much faster than the boys. During the interview a boy said that he found the experiment fun at first, but it became a little repetitive on the last day. He hoped that they could put more gadgets on the robot, for example a web-cam. When asked to describe what they did with their Lego-robot the girls answered that they solved assignment 1 and 2. For most of the girls this was just another school exercise. One group, when asked to describe what the robot consisted of, misunderstood and thought that they were to explain how the physical robot worked. The two girls on that particular group pushed the boy to do this explanation. But when they found out that it was the robot-program that should be explained, they were at least as active as the boy. Some of the boys enjoyed programming, and tried their best to program a good Robot, but they did not seem as smart or fast as the girls.

Pair programming

Studies have indicated that pair programming helps poor performers to retain their studies and pass the exam in introductory programming courses (McDowell et al, 2002; Herzog, 2005). The pupils in these experiments were organized in triplets, partly due to three being less vulnerable if one drops out, and partly due to practical circumstances like available space and Lego sets. The principle of shifting being the driver was mildly emphasized, and the teachers in general did not intervene if one of the kids took control or if one played with the bricks on the floor. The latter happened in several of the 11 years groups.

In many groups of the 14 year olds, one person would be reading the API and advising and helping the one at the keyboard, while the third paid little attention. This seemed like a productive way of working together, but little is known of its effects on learning.

In one group consisting of two girls and a boy, the boy would try hard to understand the program, and the girls would just chat about other stuff and help occasionally when the boy was stuck. While this case may correspond to prejudices of the genders, it also shows that some of the back seaters picked up the issues faster than the drivers.

The relaxed, triplets programming demonstrate that the children find their ways of working based on their interests and skills, to some extent taking notice of the instructions given. While some girls socialized and some boys played with the bricks, the majority found their ways of learning the main topic of the sessions. Pair programming seems to be too rigid for studying at this age.

Graphical and textual programming

When working with Robolab, the children had a tendency to create a huge program that was difficult to understand, before testing the program. When they finally did try to transfer the program to the RCX, they received lots of error messages and finding the errors was difficult. When using KarelJ and BlueJ, the children were much better at writing small portions of the program before testing it. When an error occurred, finding its location in the program code was easier. One reason for this change of behaviour could be that they had learnt from the experience with Robolab. Another explanation is that writing the actual program when you just use text in an editor is less fun than pasting and moving graphical icons on a computer desktop. And since the writing is less fun the children are more curious to see whether they programmed correctly so far.

Threads and gotos

The 11 year old pupils struggled both with the syntax and semantics of the Robolab Inventor programming tool.

The icons have 4 corners. One in, one out, and one for values and one last for other options. So when you wanted to sow 2 icons together you had to make sure you connected the right corners with the sowing thread, and the icons did not display the functionality of their corners. Finding errors when connecting to the wrong corner proved difficult for the kids.

When the size of the programs extended one screen, the program structure resembled spaghetti, and neither the pupils nor the teacher were able to mend them. The Robolab language enables program structures like the following

```
while (…) {
   …
   label:…
   …
}
…
goto label;
```

So even small programs could be unintelligible.

Nuts and bolts

While the Lego toys trigger enthusiasm amongst the younger kids and also supports a program execution that can easily be experienced at all age, mechanical problems detracted attention away from programming for long periods. For example, when programming a motor to run faster than the other in order to get a car to turn, the car did not turn due to the friction between the rubber tyres and the floor. The kids had to learn how to work around this, eg, by equipping the car with a belt. Learning to build a sturdy construction and use cog wheels also consumed time. The pupils thus acquired other skills than programming when working with the Lego robots, which also may be valuable learning.

Object-oriented skills

When learning new concepts, using them to refer to the things in the world which they are intended to denote is a symptom of mastery. Also, distinguishing between concepts that are similar is an important skill.

Getting the objects in the right place

The following discussion from the Java Lego programming starts with a child raising her hand and asks the tutor how they can make the robot write something on the display:

Child1: How can we make it write name?
Tutor: Write?
Child1: Name
Child2: I want to write something on the display
Tutor: Aha, write on the display

The tutor then uses their printout of the API and tells the children to use Display.

Child3: Class Display
Tutor: You have to find out where the Display belongs
Child3: Is it here? (points the mouse to the top of the Robot class)
Tutor: Where does the Display belong....it belongs to the yellow box..so....
Child3: Ok...here then (and points to the top of the Control class.)

At the top of their control class they have already declared and made a new Speaker object. This is not correct, since the speaker also is a part of the robot and not of the control.

Tutor: Sounds good, it was here you made the Speaker
Child2: What shall we call it?
Tutor: You have to make a new Display first
Child3: How do we do that?
Tutor: Look at how you did it with the Speaker

And it ended up with the children placing it at the same place where they had placed the Speaker. In the same groups program the motors and the sensors were placed correctly in the Robot class.

Subject – predicate - object

We found in many of the instances that the children had problems understanding the difference between the dot and the parentheses when they called a method with a parameter value. They would write:

```
leftMotor.forward.rightMotor.getSpeed
```

instead of

```
leftMotor.forward(rightMotor.getSpeed())
```

They had not been taught that such method calls have the same structure and semantics as the simple grammatical construction of subject – predicate – object, and that the grammatical object has to be enclosed in parenthesis in Java. Even if this had been explained, their knowledge of grammar of their mother tongue might not have been good enough to constitute a foundation for learning the programming grammar.

Dots and the genitive ‘s’

There were two different ways of talking when the children called the methods. Along with their writing they said either ‘period’or ‘s.’

This following conversation starts when the group has made an if-test that checks whether one of the light sensors sees a black tape. They return from the testing and starts writing the test for the other one.

Boy: volvo period mummi period luminosity 45
(writes `volvo.mummi.luminosity`)

…

Girl1: leftMotor stop

Boy: the boy types `volvo.leftMotor.stop()`

It ends with one of the tutors coming over to the group and helps them with the rest of the program. More of the groups where the pupils said ‘s’ as in “volvo’s leftMotor’s forward” completed the exercise.

Using the genitive ‘s’ is a sign of understanding the semantics of the dot notation.

Object, class and reference

The groups were also asked whether they could have more that one robot. The answer was yes, but these could not have the same name. The children and the teachers used the term “name” as the name of the pointer to the actual robot-object, and not a name-variable in the robot-object. They also knew about the control-class as the start of the program, also visible in Figur 4, where the control-class doesn't have a name, and doesn't have a pointer to it from anywhere.

The 14 year olds seem to distinguish between class and object, but they have not learnt that references and objects are different kinds of entities. This distinction was not taught, in order to reduce the number of concepts to be absorbed.

Without being aware of the distinction between objects and references, the pupils’ struggling with declaration of references should not come as a surprise. Declaring

```
LegoMotor leftMotor
```

and then generating and assigning

```
leftMotor = new LegoMotor(A)
```

seemed like saying the same twice and remained unintelligible.

Navigating to the objects

During the restaurant assignment, one of the groups of 14 year olds asked a tutor how they were supposed to use the `menu` object. In order to explain it, the tutor pointed to an error in their code. The group had written the following line in their control class code:

```
objecta.butler.changeNumberOfSeats(table4)
```

The correct answer would have been:

```
objecta.butler.changeNumberOfSeats(objecta.table4)
```

The following conversation took place between the girls who wrote the incorrect version and the tutor:

Tutor:	Why have you written: objecta.butler ?
Girl1:	because we thought that.....
Tutor:	I'm not saying that what you have written is wrong'
Girl2:	Because it is the butler to the....
Girl1:	because it is the butler at the restaurant objecta
Tutor:	Correct. Table4, where is that located?
Girl1:	At the restaurant objecta
Tutor:	So the butler belongs to the restaurant and...
Girl2:	and it is the restaurant's menu
Tutor:	Yes, but as you said, table4 is located in the restaurant
Girls:	???
Tutor:	You cannot write table4, it needs something more
Girl1:	No
Tutor:	Yes
Girl1:	Aha, so you have to type objecta here
	(points to the start of `table4` and types: `objecta.table4`)
Tutor:	Yes, you have to feed the computer with a teaspoon

These pupils managed to distinguish between the grammatical entities of subject, predicate and object in their method call. This may be due to two days training. When navigating to the Java object that had the role of the subject to carry out the action, they also managed to use the dot notation correctly, but they did not transfer their skill of navigation to the parameter. They had never seen dot notation in actual parameters before, which may explain their failure.

Another explanation is that they may have thought that when the statement was preceded by `objecta.` , this also applied to the parameter. This would have constituted a misunderstanding of the semantics of the dot notation and statements of method calls.

A third explanation is that the first path led them to the `butler`, who knew `table4`, so therefore it should not be necessary to tell him that the table was in his restaurant. Such an inference from the real world to code was evident in the next issue.

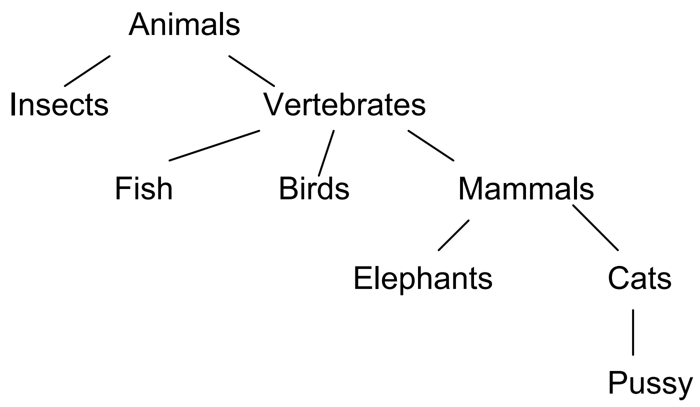
Generalization – specialization

Subclasses with Java notation was not taught to any of the classes. However, in order to assess the possibility that the children could understand this concept, a test was performed with the 11 years class.

The kids gathered in front of a blackboard in a room without computers. A teacher explained during 5 minutes the biological hierarchy of animal species and draw Figur 3 on the blackboard, emphasizing that these relations were between types of animals, except the lowest level, where individual animals were present. The children were then requested to extend the hierarchy, and more than half of them eagerly suggested animals like wasp, salmon, parrot, their pet parrot Nina, dogs, their dog Snoopy, and more pets, all placed correctly in the diagram. This extension took more than 5 minutes and was halted by the teacher.

The task given them afterwards was to draw a similar diagram of the Lego robots that the groups had constructed. The teacher mentioned that there were cars, some with belts, and that one group had made a machine for sorting bricks according to their colour.

They worked individually for about 10 minutes, and then their drawings were collected.



Figur 3. Generalization hierarchy presented to the children

Out of 27 sketches, 15 were generalization hierarchies of robots, three of which had errors. There were six aggregation hierarchies of the type Robot consists of Motor, Wheels, Sensors, etc., and six that were of less obvious order.

Half of the class of 11-12 year old kids were thus able to create their own hierarchy of subclasses of entities that they had not categorised like that before. These kids have the necessary general understanding of the relationship that is needed in order to utilize the subclass mechanism in OO languages.

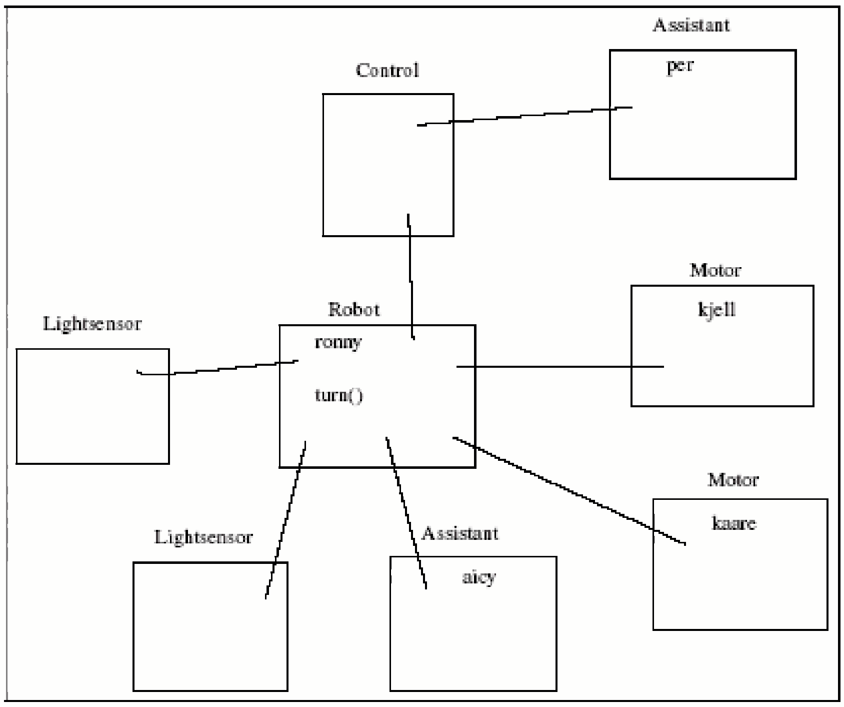
When switching to KarelJ at a later day, the kids had to modify a subclass of `class ur_Robot`. However, time was running short, so the children were not told about the Java concept of subclasses or the relation with the generalization hierarchy that they had made previously. No conclusion regarding the children's understanding of subclasses can therefore be drawn.

Reality, models and code

The pupils who were working with Lego toys mostly had a limited and straight forward domain of reality to model in their programs. In addition to the objects that represented motors, sensors, etc, there were two abstract objects in their programs that originated in the programming environment. An object of `Class Control` was the start of the program, which was described as a control-center for controlling the world. This is similar to the main-method, but without the Java `static` notation. There was also an `Assistant` object for hiding thread-programming that has to be included in order to control execution.

When asked to draw a model of the robot program, some of the 14 year olds came up with the model in Figur 4. The `Control` and the `Assistant` are included, and an additional assistant was also included.

Another group was asked what the robot consisted of, and asked to draw this on the paper. A member of the group drew a motor-object and attached it to the robot-object. When asked whether the robot only consisted of one motor, another participant in the group drew the second motor on the model. The robot also consisted of two touch-sensors and one light-sensor. However, the children did not make object-boxes like they did with the motors, but instead just wrote the sensors directly into the robot-object. None of the other groups did the same.



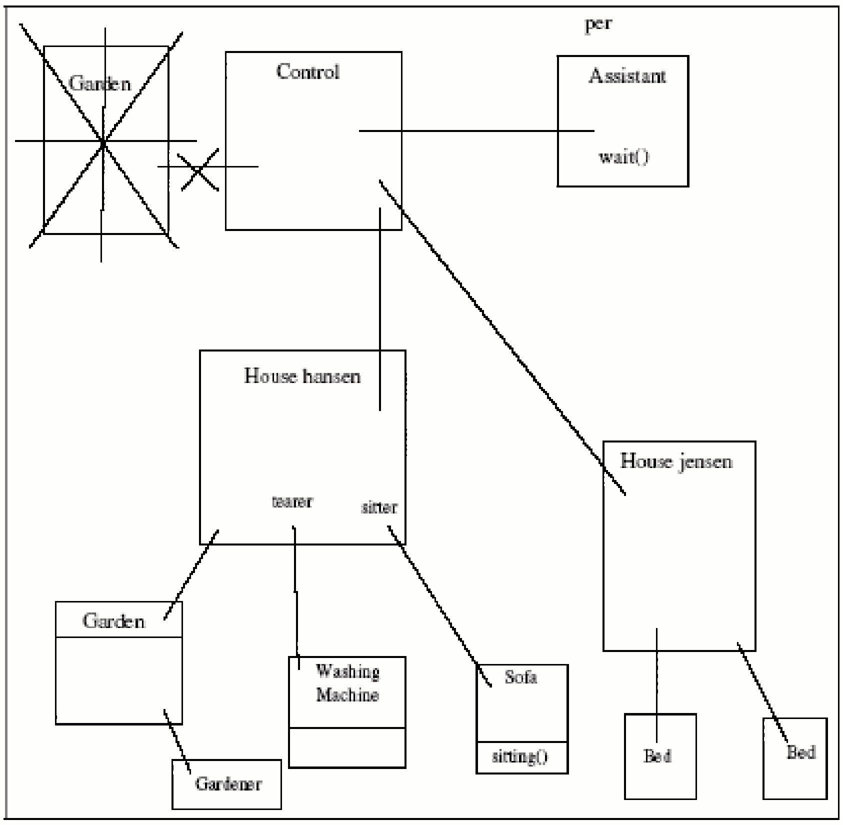
Figur 4. Model of the robot

The robot case had a reality immediately available for experience, the code and the model drawn by the pupils. In the restaurant case, the reality had to be imagined, based on the children's prior experience, while the code was retained. In the last exercise, also the code was omitted, and this time the kids were asked to make a model of a house.

One of the groups made the model reproduced in Figur 5. Without instruction, they included the Assistant and Control. They were instructed to make a garden, And looking at the drawing the children have to tell the gardener to cleanup for a number of seconds. The method is called from the control-object. Their first try at the long call sentence was: `control.househansen.garden.gardener.cleanup(1000)`. However, you don't have to type `control.` when the call is made from the control-object. This mistake was not done so much in the coding of the robot, but without the programming environment of the robots, the kids seem to place themselves outside the model and interact with it through the `control` object.

Some of the girls feel that the call-sentence is unnecessary long. They claim that since the garden is such a big part of the house, the program already knows that the gardener is located in the garden. They may assume that the program already has a representation of the real world that is similar to their own.

Concerning the garden, one can argue that since it is outside the house, it should not have been modelled as included in the house as the same way as the sofa. If they were to follow the logic of inclusion consistently, they could have included the house in the garden and not vice versa.



Figur 5. A model of a house.

Discussion and conclusion

Based on the observations of the different topics of mastery above, stating that some kids understand object-orientation would be a sweeping statement that would have to be modified. What we see is rather that the competence of OO programming is constructed by a number of skills of using the different OO constructs in ways that make the programs perform and that the teacher finds acceptable. Also, the pupils have to be able to relate code to its execution and to the real world being represented.

In addition to the technical skills, the students have to acquire the social competence needed for programming, which at this introductory stage includes commenting peers' performance, helping others and interacting with the tutors.

Comparing with Borge (2004), the issue of class, object and reference is similar and triggered by the way these concepts are taught rather than the age of the learners. Détienne (2002, p65) mentions that novices have a tendency to use instances of classes without having generated them, which corresponds to the kids using references to objects without having declared the. The subject-predicate-object confusion has not been reported earlier, but the authors have seen this mistake amongst novice university students during teaching introductory programming. No earlier documentation of the navigation problems have been found either, but in general assuming that the computer knows more than you have told it is a common "superbug" by novices (Pea, 1986). The question of where to place a class in the code or an object in a model is also well known amongst adults, and also experts have to struggle with such design issues.

Kaasbøll et al (2004) noted that learners who in their programs have to represent a real world domain not being present for experience have a much higher cognitive load than if programming visible objects. The children's quick learning of the classes of Motor, Sensor, etc, corroborates this hypothesis.

The Robolab and the Lejos programming tools for Lego robots enabled learning programming in a setting where the execution of the program is visible in space, which eases the experience of how the program works, as compared to imagining methods being called in the RAM of the computer. However, Robolab deserves improved language mechanisms for structuring the imperatives, and Lejos can be improved through hiding threads, exceptions, `Static` and `Main`.

The restaurant simulator was useful for programming a real world domain before having learnt to identify classes, thereby providing an intermediate learning step from the toy worlds of Lego and Karel to solving programming problems based only on a natural language description.

In total, the learning issues presented in studies of adult learning differ partly from those reported here, but the issues seem to be generated by the topics being taught rather than the age of the learners.

Nygaard's claims that OO is a natural way of conceiving the world seem to be confirmed when the domain of the program is open for direct experience, like Lego Mindstorms robots and KarelJ (Borge, Fjuk and Groven, 2005), and the programming environment offer some predefined classes.

Vessey and Conger (1994) report that OO is more difficult to learn than other paradigms, and their conclusion is supported by Détienne (1997).This result seems too general when considering the different pedagogical approaches possible. The OO concepts can be learnt by kids, but Détienne's conclusion that identifying classes when analysing real world domains has not been refuted by the experiments reported here.

Implications for teaching

Teaching by means of Lego requires that one or two days additional time for learning the mechanics has to be scheduled. Three to four days of training can be recommended for kids aged 11-12 working with Mindstorms and Robolab. Older children, and girls in particular, may find Lego less motivating.

The experiment demonstrated that children at the age of 14 can learn object-oriented programming, provided an environment that supports immediate experience of program execution. Teaching OO programming in mandatory school at this age seems therefore feasible. Considering that many kids at that age struggle with abstraction, only a portion of the pupils can be expected to be able to transfer the skill of programming robots to other domains.

Teenagers in high schools have more training in formal theories, and the less theoretically able have chosen other educational paths. Therefore, more profound programming competence can be developed in high schools.

The argument that OO is superior due to its strength in modelling the real world has been put forward as the main reason for an OO first (Groven, Hegna and Smørdal, 2003). Even though modelling the world is essential in programming competence, starting out by modelling the real world adds complexity to the learning environment. The teaching should rather start with programming hardware or software robots or similar visible domains, and when the students have developed competence in that limited environment, they can move on to real world modelling.

References

Borge, Richard Edvin (2004) *Teaching OOP using graphical programming environments : an expermental study.* Master thesis, Department of Informatics, University of Oslo

Borge, Fjuk and Groven (2005) *Using KarelJ collaboratively to facilitate object-oriented learning.* This volume

Détienne, F., (1997). Assessing the cognitive consequences of the object-oriented approach: a survey of empirical research on object-oriented design by individuals and teams. *Interacting with Computers*. 9, 47-72.

Groven, Arne-Kristian; Håvard Hegna; Ole Smørdal (2003) OO learning, a modeling approach. In Luca Cardelli (ed.) *ECOOP 2003--Object-oriented programming: 17th European conference – proceeding*s. Darmstadt, Germany

Hegna, Håvard (2003) *Using Active Objects in a "Model First" Approach to the teaching of Object-Orientation.* COOL seminar, Norwegian Computing Centre, Oct.

Herzog, Christian (2005) *Learning programming in pairs.* Master thesis, Department of Informatics, University of Oslo

Kaasbøll, Jens; Ola Berge; Richard Edvin Borge; Annita Fjuk; Christian Holmboe; Terje Samuelsen (2004) *Learning Object-Oriented Programming.* In E Dunican (ed.) Proceedings of the Psychology of Programming Interest Group 2004. Carlow Institute of Technology, Ireland

Lego Mindstorms (2005) http://mindstorms.lego.com

Lego Robolab (2005) http://www.ni.com/company/robolab.htm

Lejos. Java for the RCX (2005) http://lejos.sourceforge.net/index.html

McDowell, Charlie; Linda Werner, Heather Bullock and Julian Fernald (2002) The Effects of Pair-Programming on Performance in an Introductory Programming Course. *SIGCSE Bulletin* 34, 1, 2002, 38-42

Papert, Seymour (1980) *Mindstorms: Children, Computers and Powerful Ideas.* New York, Basic Books

Pea, Roy (1986) Language-independent conceptual ”bugs” in novice programming. *Journal of Educational Computing Resarch*, 2, 1, 1986

Vessey, I., & Conger, S. A., (1994). Requirement Specification: Learning Object, Process, and Data Methodologies. *Communications of the ACM.* 37(5), 102-113.

Teaching the object-oriented concept Encapsulation to 14 year old children using Lego® Mindstorms™ and the problems with this kind of teaching method

[A Case Study]

Roar Granerud
Institute for Informatics
Blindern
Oslo, Norway
rgraneru@ifi.uio.no

Ruth Merethe Evang
Institute for Informatics
Blindern
Oslo, Norway
ruthe@ifi.uio.no

Marte Oedegaard
Institute for Informatics
Blindern
Oslo, Norway
marteod@ifi.uio.no

ABSTRACT

During three days at a junior high school in Norway, some students got the chance to play with Lego® and write programs using Java^{TM}. They used a modified version of a Java^{TM} API called lejos[1]. We wanted to see if they could understand some of the object-oriented concepts. During the sessions, groups of three were videotaped and at the same time their computer-screens were recorded. On the last day the Lego® robots were exchanged with a restaurant simulation to see if the knowledge received from the robot-programming could be transferred. The last day four groups were interviewed in front of a camera and asked how much they had learned. The main object-oriented concept we tried to teach the subjects was encapsulation. We wanted the subjects to get a sense that certain objects "belonged" together, and that these objects were responsible for their parts of the program. We wanted them to go above the computer program and get a more general understanding of what they were doing. They would learn how to access the object and the method they wanted using Java^{TM} dot-notation.

1. INTRODUCTION

The COOL-group, formed by Kristen Nygaard in 2001 has been researching different ways of teaching object-orientation. Last year a study was performed by the COOL-group. A handful of CS1[1] students programmed virtual robots on the computer screen using Karel-J, a small Java^{TM} -API for programming 2D robots on a grid-based screen. The idea with this study was to see if a graphical representation of the students code would help them understand the object-oriented concepts encapsulation and inheritance. Some of these subjects were also participants during this case study. Part of the conclusion from that study was that graphical representation of objects helps students understand what objects really are. Françoise Détienne speaks about the naturalness of object-oriented design[4]. So when someone is programming a physical object like a Lego® robot, it would be natural to use object-oriented design and programming. The object-oriented knowledge gained by these experiences would then prove helpful in other object-oriented programming tasks. Is object-oriented design as natural as Détienne says that it is?

According to the late Norwegian professor Kristen Nygaard "Teaching object-orientation must start with at sufficiently complex example [6]" In our experiment we started with a simple example, but after two days we totally changed the domain from physical Lego® robots to a simulation of a large restaurant. We still didn't use a procedure first way of teaching object-oriented concepts. We didn't use much time explaining loops and if-sentences, our goal being to teach what objects are in the most understandable way.

In Nygaard's restaurant there are lots of objects that interact, and therefore the natural way of simulating this restaurant would be using an Object Oriented approach [6] . The objects in the restaurant are customers, waiter, tables and the food plus many more. Hopefully the object-oriented knowledge gained from programming Lego® robots would prove useful when trying to solve another vast assignment containing many objects. Would the subjects be able to make the abstraction?

We hope to find out if the subjects will be able to understand what they are programming with just three days of training, and with no prior knowledge of either Java^{TM} or programming. Will the subjects be able to use the knowledge gained from this project to design programs in an object-oriented manner? We also want to know if the use of Control Technology (see chapter 2 for an explanation) help the subjects to understand their programming better. Will they see the connection between the robot programs and the physical

[1]CS1 is short for Computer Science 1. This is the first computer course taught at universities and colleges. It usually involves computer programming in a chosen language.

robot they make?

2. PEDAGOGICAL INVENTION

We wanted to use Lego® Mindstorms^{TM} as the mean for teaching object-orientation. With the use of a custom made Java^{TM} API called Lejos we wanted to use Control Technology[3] to see if a physical representation of an object helps in understanding how the Java^{TM} -representation works. If the subjects in their Java^{TM} -program writes
"robot.rightMotor.forward()",
they will be able to see both the physical robot, and the attached physical motor. Due to this it will be easier to see why it is natural in a computer program that both the data and the methods, that is associated with an object, is hidden inside that very object.

Lego® has been used as Control Technology in a lot of other subjects, such as physics and math[3]. Lego® Mindstorms^{TM} is really just an advanced version of the turtles in Logo[7]. Logo has been used to teach geometry and other math-aspects especially to children and young adults. It has also been used to some programming aspects i.e. recursion[7]. According to our knowledge this is the first time that Lego® has been used to teach object-oriented concepts. And if using Lego® is a good way of teaching object-oriented concepts, this will hopefully be shown in the final exercise we had the last day: The object-oriented restaurant.

3. BACKGROUND

Studies show that students need to understand both the low level programming model and the high level domain model to be good programmers. In object-oriented models we focus on the domain knowledge[8].

Last year there was a study of graphical environments for supporting the teaching of object oriented programming and with a main focus on Java^{TM}. [2] The experiment concluded that using a graphical environment as a support helped the students greatly in learning the first step in object oriented programming.

We wanted to examine this further, and tried to find out if it is easier to learn object oriented programming with physical objects. Our idea was to give the students a visual contact with their program. When they programmed their robots, they saw how it worked as the robots did different things.

In our experiment we were trying to teach the students by giving them Lego® Mindstorms^{TM} robots to build and program them with Lejos. Lejos is a Java^{TM} library made for Lego® -robot programming. Since Java^{TM} has a complex syntax we gave them a simplified template to start with. They only had to copy and paste in the correct places in order to make the robot do simple things like turning, running forward and backward. In order to use the sensors they had to read the API they were given.

In this study our focus was on two object oriented concepts used to reduce complexity, abstraction and encapsulation. Our goal was to make the students learn these concepts. We started the projects with more concepts in mind, like subclasses, we found no good way of using the Lego^{TM} to teach that concept.

Abstraction is the process of picking out common features of objects and procedures. You simplify the program by picking out relevant information, and not using all available data. Abstraction also occurs in non-object oriented programming, like C. In C they are called structs. Abstraction leads to encapsulation and information hiding.

Encapsulation is the process of combining elements to create a new entity, like procedures, methods or objects. One aspect of encapsulation is information hiding. This gives the opportunity to hide certain data and prevent these data from changing by accident, by not allowing objects to access other objects variables directly.

There are two common ways of learning, behaviorism and constructivism. Behaviorism is the learning based on stimulus. A different path in behaviorism says that people can learn a certain behavior through how a persons actions are rewarded, either through positive or negative reinforcement. [5]

In constructivism, the knowledge is not ready made, but is constructed in a person's psyche through the interaction with the environment around them.[2] We wanted the subjects to build their cognitive schemas in this way by seeing the Lego^{TM} robot as an object interacting with the environment using its sensors.

4. METHOD

The study's purpose was to survey and explore the way people learn and practice object oriented programming with a custom made Java^{TM} API called Lejos. The study evaluates the subjects's behavior and quality of product, but is not intended to result as a programming template, nor a programming guide. However we are doing experimental studies with different subjects and new learning styles. We hope the answer would aid us in understanding of teaching object oriented concepts and show us that young adults also enjoy programming in object oriented languages.

Programming Lego® Robots using an Object Oriented programming language like Java^{TM}, 14 year old children were introduced both to programming and object-oriented concepts. Using Control Technology[3] the subjects were able so see in real life what their programming were doing to the Lego ®Robots.

The students were given a simple API describing what their Lego® Robot could do, and how to do it. They also received a program template with a simple working Java^{TM} program. We wanted to see if the subcjets were able to place the correct objects (or pointers to objects) in the correct objects. In other words, we wanted to see if we could teach these subjects the use and power of encapsulation which is one of the most important object-oriented concepts. An object knows about what is relevant for that particular object, and if other objects wants to access some of that information, they have to go through that object (using the correct dot-notation).

The third day after programming Lego® robots, the subjects were presented with a new task. This assignment was to program a simulation of an object-oriented restaurant. The subjects were given a restaurant-template where the different restaurant-classes already were made. In order to succeed, the subjects had to use an object with the another object in the correct way. They could make customers, waiters, chefs and tables. There was also a menu-object in the restaurant. The task was to make the waiter seat all the guests, take their orders and making sure that the guest got what they had ordered. The chef cooked everything that was ordered in one go. The results from this assigment would show if the subjects had learned anything, both of Java^{TM} -programming and the object-oriented concept of encapsulation and sharing of responsibility.

We used qualitative research methods. Qualitative research methods were originally developed in the social sciences to enable researchers to study social and cultural phenomena [2]. Different methodology is used in different part of science, we believe that the study lays closer to the soft aspect of science, also the closest to social science.

Qualitative data sources include observation and participant observation, interviews and questionnaires, documents and texts, and the researcher's impressions and reactions[2]. The observational studies consisted of recording video, both covering the subjects when they programmed alone and with assistant. The recorded material existed of sound and computer screen of the students' computers, while they programmed. The study ended with some students being handpicked for an interview. Four groups of three subjects were being questioned the final day.

The subjects were about 30 children. They were thirteen to fourteen years old. They had no prior experience in programming. The experiment started with a short introduction where they were taught some object oriented concepts and guidance to use the Java^{TM} API called Lejos. We wanted to throw the subject into complex problem solving, without a long pre-lecture.

Video cameras were used during the experiment. Both the computer-screen and the subjects working together in the groups were filmed simultaneously. We would then be able to see what the subjects were doing in their programing and simultaneously see the subjects's discussions. This would be very helpful when trying to determine if the subjects understood what they were doing.

In the closing stages of the experiment, the subjects were given small assignments related to object orientation. This was suppose to map the subjects deeper understanding of object-oriented thinking. They were given papers on which they were going to sketch the different objects and inscribe sentences with the terminology they were introduced to earlier.

5. RESULTS

5.1 Lejos

The main object-oriented concepts that we wanted to teach the subjects encapsulation. We hoped that they would be able to place the objects in the "correct objects", and logical methods in different objects.

The subjects were able to understand that the motor belonged in the robot-object. They were also able to use the pre-made methods in the motors and sensors from the control-object. When wanting to turn the robot around they would write the turning sequence in the control-object step by step, and if they needed the turning sequence again, they would copy and paste the sequence they had already made. We wanted them to understand that a better solution would be to create a "turn" method in the robot. We didn't use much time explaining what a method was and how to use it. We hoped that they would see the correlation between the methods in the sensors and motors, and methods that they could create themselves. Unfortunately it can't be said that any of these results are conclusive. The reason for this is probably because using a copy and paste technique takes much less time and is easier than writing a method, something that was new to them. Another reason was that the main goal for the subjects was to complete the exercise, and just making the program run with the correct results.

We were aware that there would be a limit to the amount of new things learned. We only had three days, and much time was wasted playing with the cameras and the Lego® . But we still feel that we contributed to the subjects's understanding on how robots in the real worlds are programmed, and how to model a real world object into an object-oriented program. When asked about everyday robots the subjects knew about, they mentioned a cleaning robot that used motors and different sensors to clean most of the floor. The subjects were now able to understand how this sort of robot were programmed.

5.2 Restaurant

After programming the physical Lego® robots, we hoped that the subjects had gotten a sense of ownership. That the different objects belonged to another object. In their programming they would write:

```
......changeNumberOfSeats(table4, 4)
```

instead of telling the program that it was restaurant.table4 that needed then number of seats changed. This was an error that was done by a lot of the groups and it shows that ownership and belonging isn't very natural, even after working with physical objects where the ownership can be seen with your own eyes. After being asked if the table belonged somewhere (a very leading question) they answered that it was part of the restaurant and was able to fix the error. This shows that the subjects had some difficulties finding out how the objects are interconnected by themselves, but when they understood how they were connected, they programmed it correctly in Java^{TM}.

In a discussion on how to place the different persons at a table, the subjects wondered how to use the waiter-method: placeByTable(Table, Person). Their idea was that some of the guest entered the restaurant at the same time, and since they should be seated together, the waiter should be able to show them all to the table at the same time. So instead of writing:

```
restaurant.waiter.placeByTable(restaurant.table4, ole);
restaurant.waiter.placeByTable(restaurant.table4, per);
restaurant.waiter.placeByTable(restaurant.table4, kari);
```

They wanted to write:

restaurant.waiter.placeByTable(restaurant.table4, ole, per, kari);

This is a more natural way of thinking, which is what we wanted to do. They should be able to seat all the persons at once, and this is probably what they would do if they had made the placeByTable-method themselves.

The subjects didn't have logical errors only. One curiosity was when the orders were taken and it was time to cook, they would order the cook to make all the meals as many times as there were guests. And not cook it all at once. This shows that these subjects didn't really think naturally when programming the restaurant and instead used a more copy-paste kind of method. But since the only reason why the logical thing was to cook the meals all in one go, was that the cook.cookFood() method took no parameters.

Some of the groups finished this assignment very quickly and got the message saying that the courses were delivered correctly. These groups had taken the easy way out and ordered the same types of food for all their customers, so that when the waiter delivered something to a person it had to be the correct course. When we said that they had to order different courses, their program reported that the customers had gotten the wrong order. This was very discouraging, since they felt that they had completed the course, that is, getting their program to display the correct messages. It was fairly obvious that some of the subjects were very tired of programming the last day and just wanted to get it done and go home. We found this a little strange, since these subjects usually had long school-day, and they had to be at school all day no matter how fast they did their exercises.

5.3 Interviews

In an interview with four of the groups on the last day, some of the subjects were asked to draw their program on a piece of paper. The had never done this before. The point of this exercise was to see if their cognitive idea of the program was anything like the idea we wanted them to have. A simplified UML-diagram[2] consisting of two squares connected together by a line. One square was called "Control" and the other was called "Robot". In the "Control" square there was a "variable" which was the name of the Robot, i.e. volvo. They were explained that this was the pointer named volvo that pointed to a Robot object. Starting with this simple drawing we wanted to see if the subjects were able to fill in the rest of the Lego® robot program structure.

Most of the groups (3 out of 4) were able to connect at least one Motor or Lego® motor correctly on their Robot. Some of the groups failed to name the motor correctly, because they didn't understand the difference between the name of the object (the pointer to the object) and the object itself. We didn't make a lot of effort in trying to teach the subjects the difference between a class and an object. And the interviewer didn't make fuss about the pointer being called Robot or volvo[3]. The important thing was that they understood what they had to write in order to give a command both to the robot, and to its parts. And all the groups placed the motors in the robot-object and not in the control-object. And because of this fact, we believe that since they actually placed the physical motor on the physical robot this would be a logical thing to do in the computer program too. And since this drawing was a graphical representation of that very robot, they had to put it in the robot-object. As they said: 'Because it belongs there'. And this shows that the idea of using Lego® Control Technology to teach responsibility sharing and information hiding in object-oriented programming was a success.

6. DISCUSSION AND CONCLUSION

There were a lot of different results from this study. Not all of them was associated with object-oriented comprehension. They will be mentioned in this article since they are relevant for teaching in general, and also for teaching object-oriented concepts.

The first issue of this sort of exercise is that 14 year old children don't have much training in front of a camera. Not that they need it to be filmed, but the cameras were an extremely disturbing element. A common thing with most of the video sessions were that the first 15 minutes of each session were productive, and the participants had fruitful discussions and worked very well on their assignment. But suddenly one on the group would discover that the camera was pointed toward them, and the rest of the session would just consist of 14 years olds making funny faces in front of the camera. We should have trained them in being filmed a week before the actual experiment took place, but we didn't have the time. They didn't have any training in using documentation either, but it was clear that the API was much more used on the last day when programming the restaurant.

A second issue was that there were three people on each group. So what happened on some of the groups was that two people wrote the code and discussed the program, and the third subject wandered off to disturb another group. The participants were only 14 years old, so this is very understandable. People that have participated in group programming, all know that being the odd man out tends to be pretty boring. But since we had to use a whole school-class this was the only option.

Admittedly three days isn't a long time. But we still feel that they learned a lot these days. Mostly because of the good performance on the restaurant-assignment. Either the tasks were too easy and trivial, or they got too much help and scaffolding. [4] Or they actually used the knowledge gained from programming object oriented Lego® robots to understand how the object-oriented restaurant was built.

[2]UML is an object-oriented design and analysis language. Its a visual presentation of a computer system where the different parts of a computer program is drawn as squares with a name, a content and the different associations between these squares.

[3]In our template that the students started to program in, we had called the pointer to the robot: volvo. So most of the student groups called their robot volvo

[4]Scaffolding is the master-apprentice way of teaching. You help a lot at first, and gradually offer less and less assistance, until the apprentice is able to perform the task on his own. In this case we mean that we helped them too much and not giving them a chance to do it themselves.

7. REFERENCES

[1] *LEJOS. Java for the RCX (http://lejos.sourceforge.net).*

[2] R. E. Borge. Teaching oop using graphical programming environments. Master's thesis, University of Oslo, 2004.

[3] B. Denis. *Control Technology in Elementary Education*. Springer-Verlag, 1993.

[4] F. Detienne. *Software Design - Cognitive Aspects*, chapter 5. Springer, 2002. Programforselse sett p som tekstforstelse.

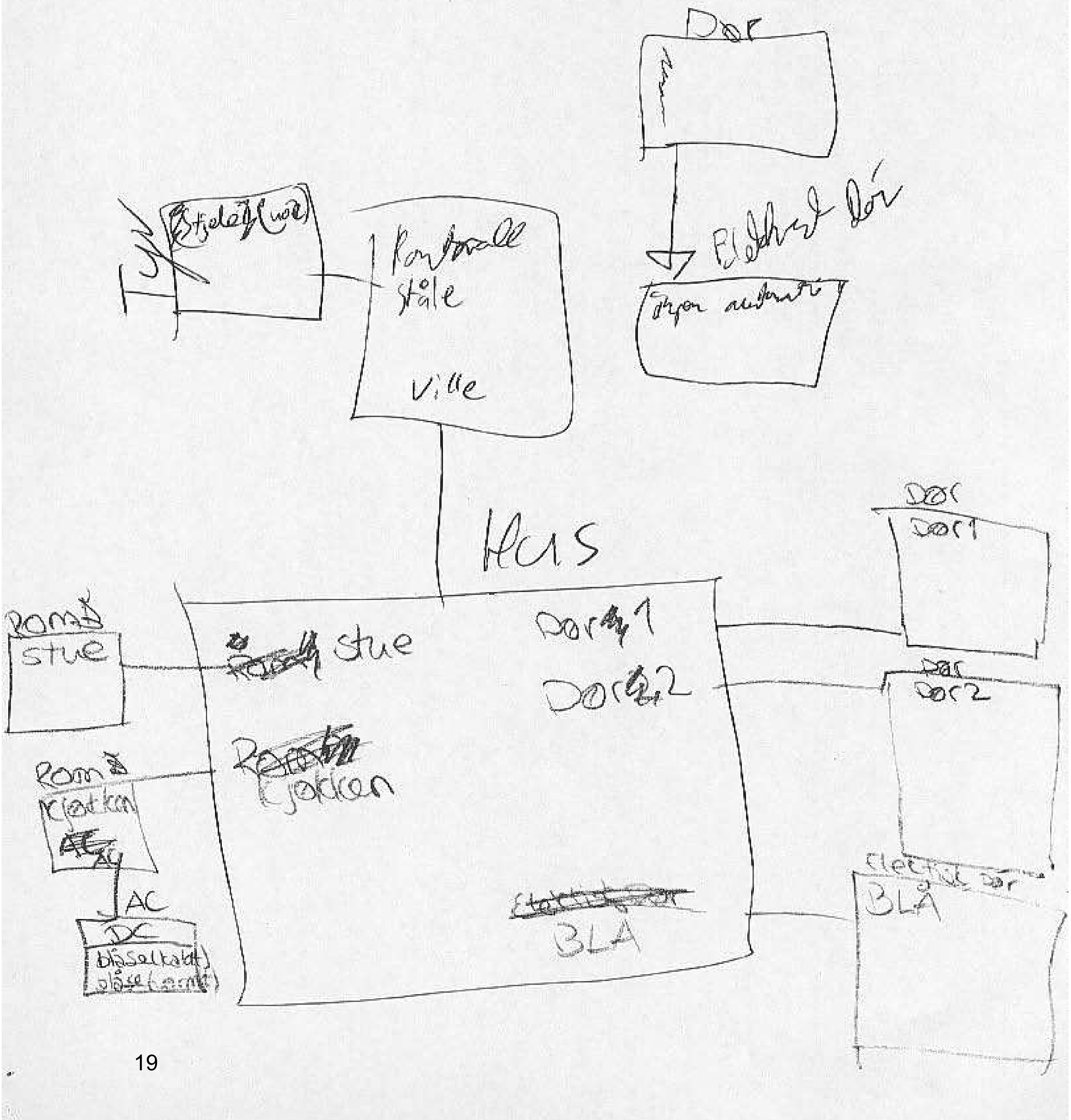
[5] J. G. Greeno and A. M. Collins. Cognition and learning. *Prentice Hall International*, 1996.

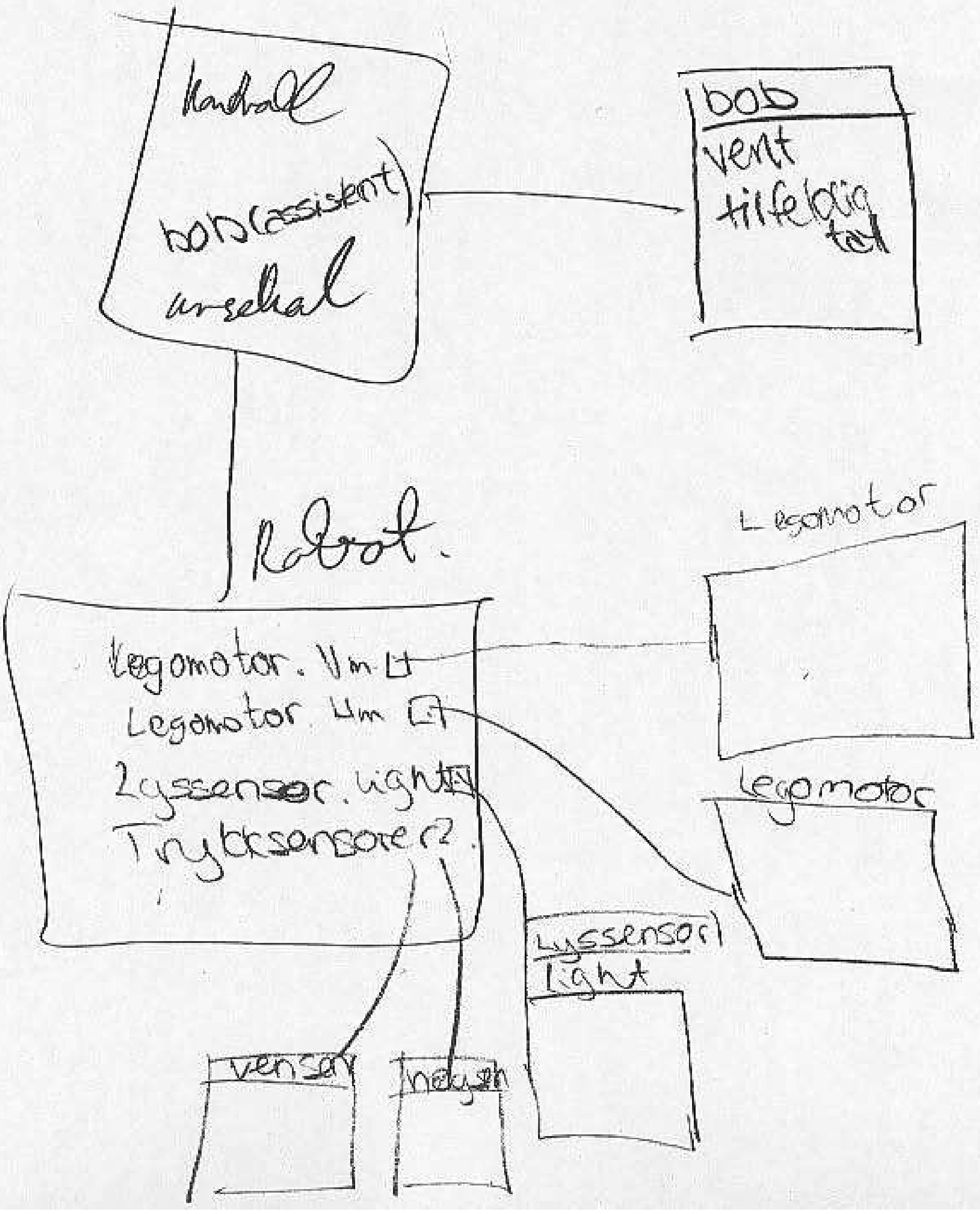
[6] K. Nygaard. The restaurant example. www.intermedia.uio.no/english/projects/cool/complex.html.

[7] S. Papert. *Mindstorms - Children, Computers, and Powerful Ideas*. BasicBooks, 1993.

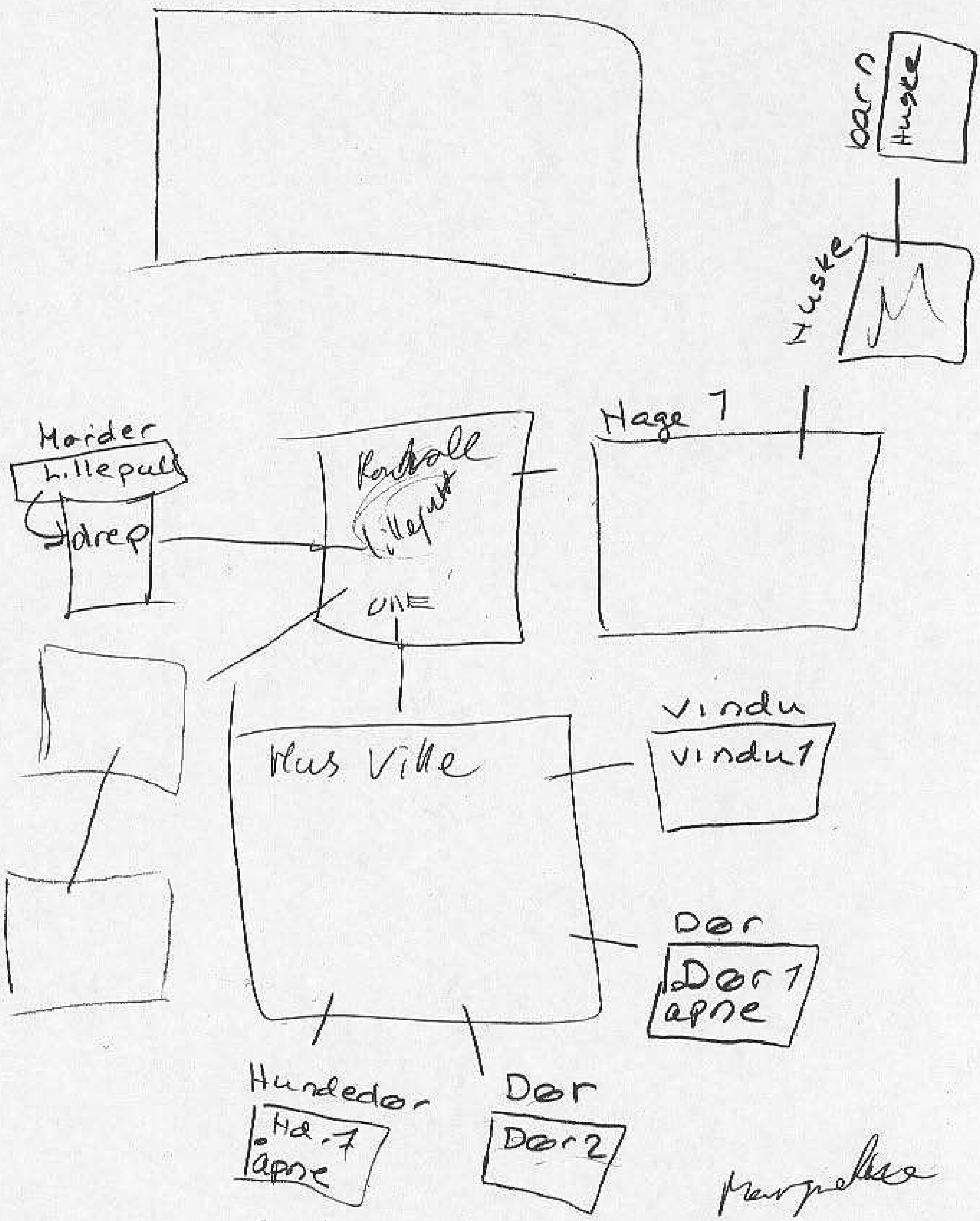
[8] S. Wiedenbech and V. Ramalingam. Novice comprehension of small programs written in the procedural and object-oriented styles. *Academic Press*, 1999.

ville . kjøkken . ~~DC~~ dc . blåse (varmt)
ståle stjele (ville kjøkken AC);

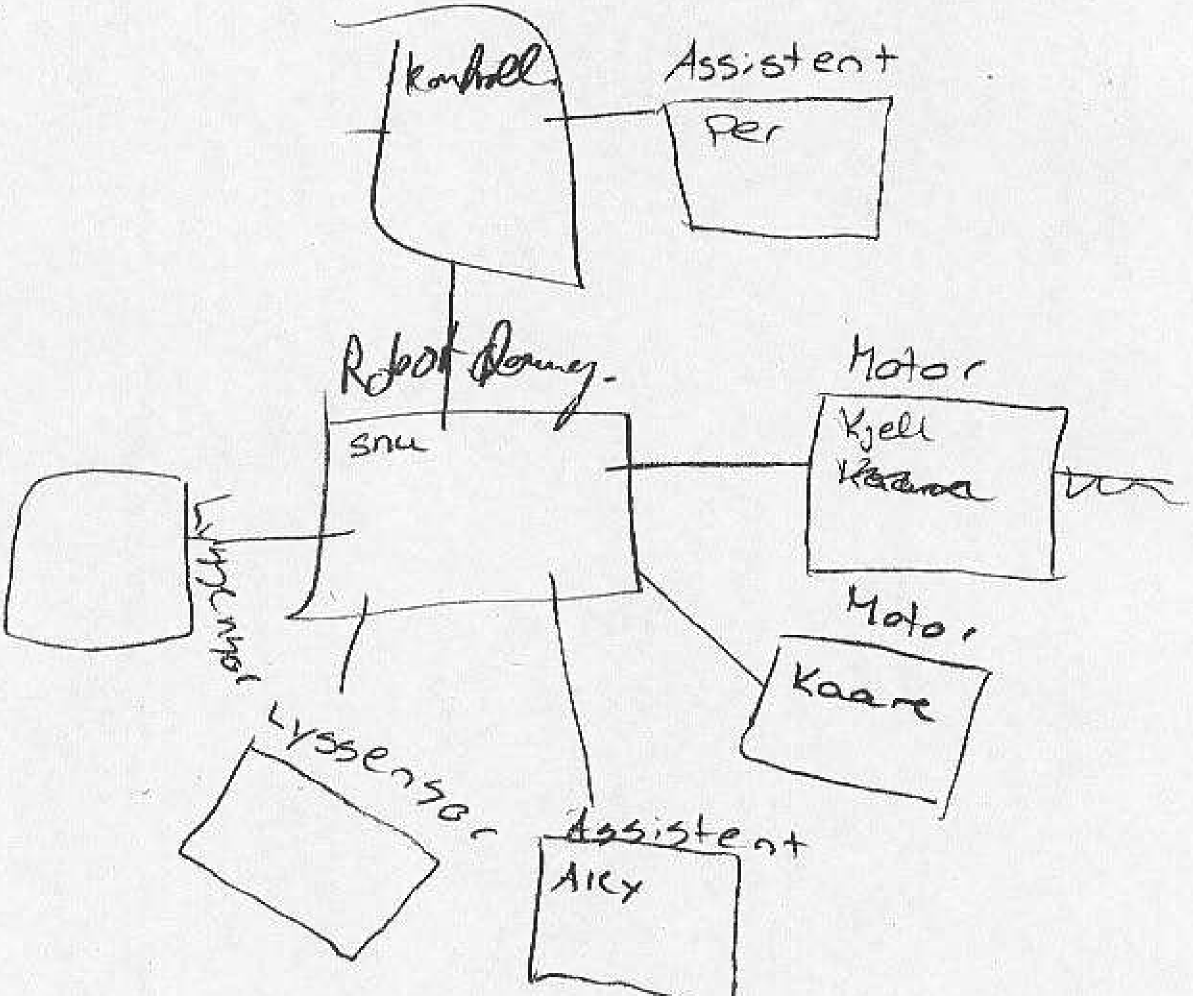



Handball
bob(assistent)
urselal
bob
vent
tilfeldig tal
Robot.
Legomotor. Um L1
Legomotor. Um L1
Lyssensor. light
Trykksensorer2
Legomotor
Legomotor
Lyssensor
light
vensar
høgyh

```
Ville.dør1.åpne;
lilleputt.drep(Hage1.Huske.barn);
```



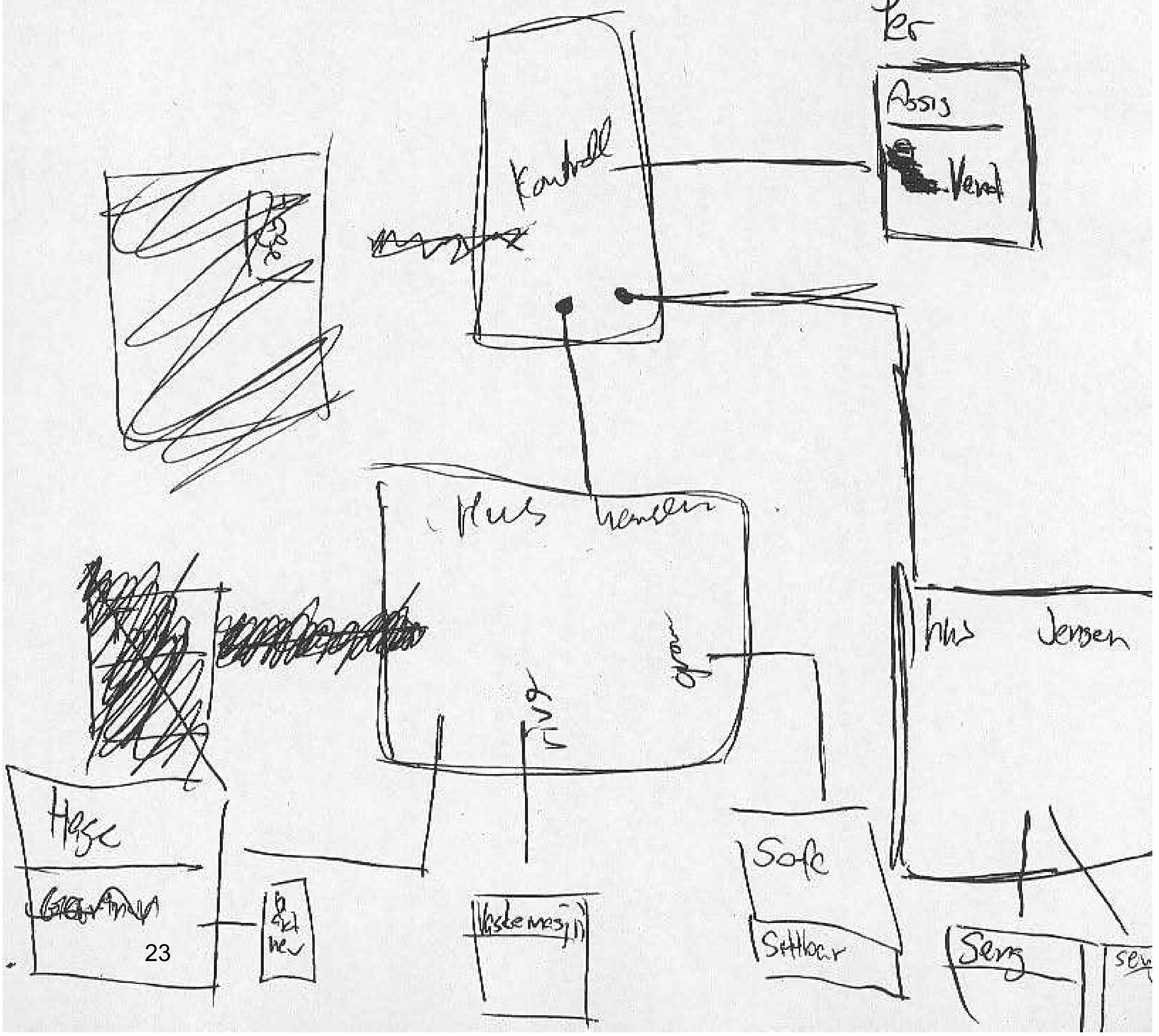
Marjukka
Rahikka
aleksanda.

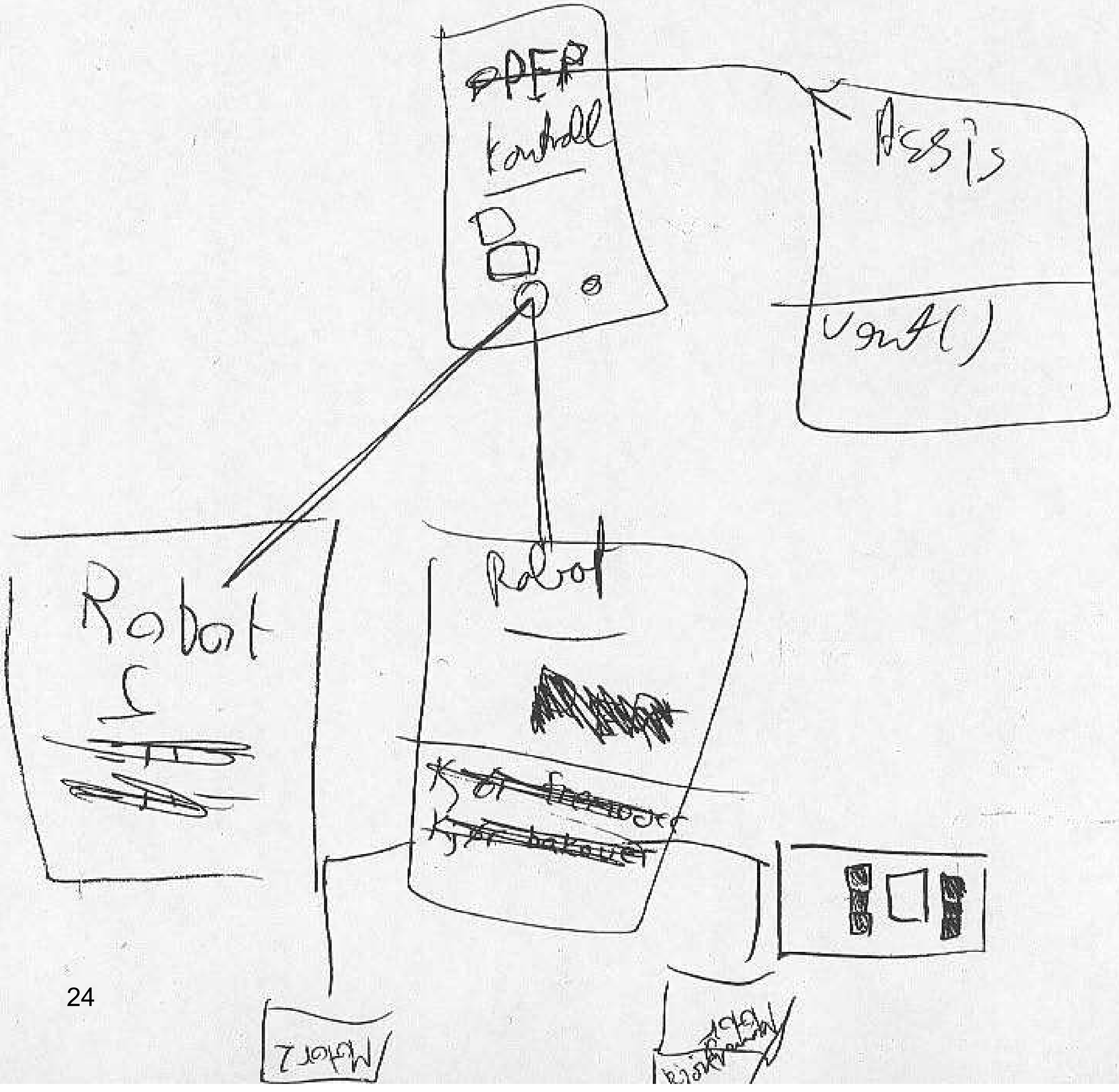

Kontroll
Assistent
Per
Robot Rammey
snu
Motor
Kjell
Kaare
Motor
Kaare
Lysmåler
Lyssensor
Assistent
Alcy

Per = new Assistent;
Tatert = new Gartner

s gane
dramatiske
jodsien.

kontroll. husHansen. hage. Gartner. gjør pent [10000].(100





~~OPEP~~
kontroll
Assis
vent()
Robot
Robot
C
~~Kjør fremover~~
~~Kjør bakover~~

~~kjøkken~~

v.

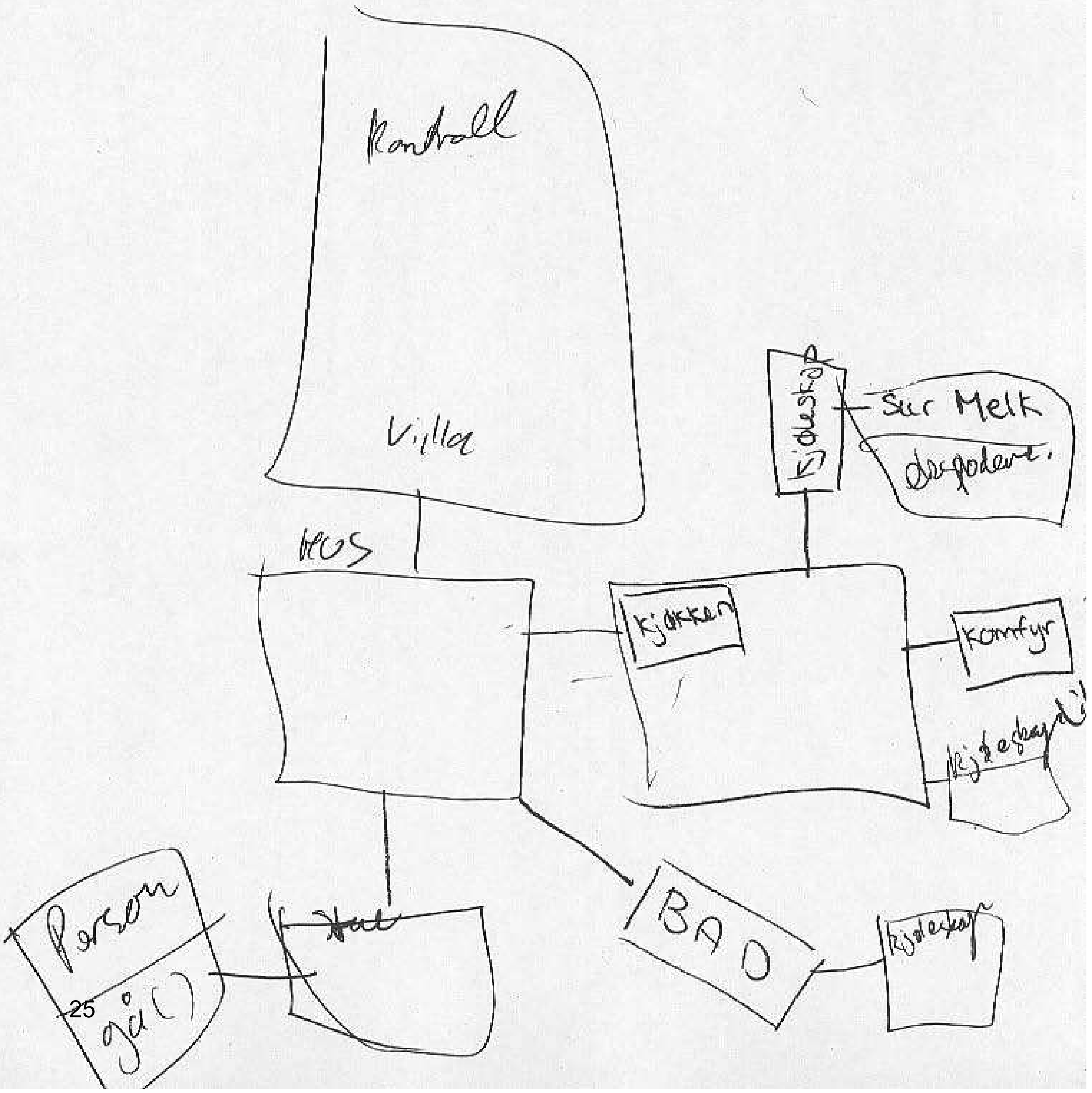
villa.kjøleskap. Melk. eksploder.

villa.stue.person.gå(villa.kjøkken.);

Fatima

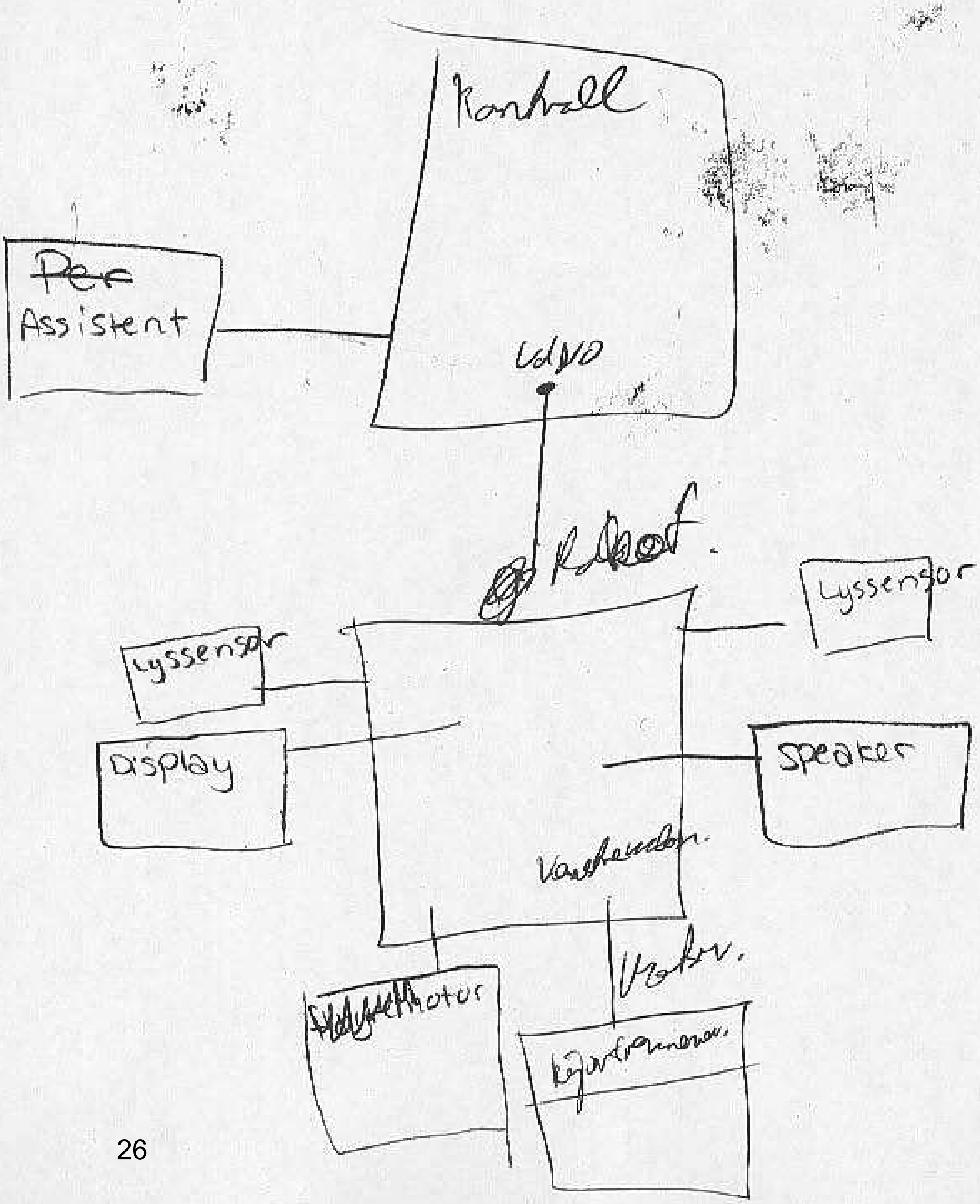
Christine

Britt helene.



/

Volvo. KøyreMotor. KjørFremover



Plan for mandag 19.april kl 0830-1400 (3 økter a 90 min + 15 min mat + 2 pauser på 15 og 30 min)

	Praktisk aktivitetsmål	**Teknologisk læringsmål**	**Informatikk/OO -læringsmål**	**aktivitet** **Praktisk gjennomføring**	**varighet**	**Ansvarlig**
1	Løse opp og komme i gang			LITEN MOROLEK	5 min	Christian
			Første møte med kategorier og egenskaper	Elevene bygger et hus hver med de klossene som er tilgjengelige. Dette er antakelig ikke så lett da brikkene ikke nødvendigvis egner seg. De må derfor gjøre kompromissløsninger og valg som har noe med sentrale egenskaper ved et hus å gjøre. Dette blir tema for ettersamtale først innad i gruppene og så i plenum	20 min	Ole forklarer aktiviteten og leder gruppesamtalen etterpå.
			Fortsette litt med kategorier og egenskaper	Rydde klossene tilbake på "plass". Veiledere initierer diskusjon i gruppene om kriteier for sorteringen. Kort oppsummering i plenum	10 min	Ole
	Avbrekk og "teambuilding"			LITEN MOROLEK	5 min	Christian
	Ha bygget en bil med motor	**Bli kjent med** de mekaniske brikkene og **forstå** mekanikken bak tannhjul og drivakslinger		Bygge en bil med motor, tannhjul og drivsystem. Ark med kontrollspørsmål som de kan jobbe seg gjennom, stille hverandre, finne svar på ved å prøve. Eks: størrelse på tannhjul. For/bakhjulstrekk vs belter. Forskjell h/v side.	50 min	Christian holder undervisningen i plenum. Roar / Richard lager ark med "lede-oppgaver".

				Rotasjonsretning på motor i forhold til forover/bakover.		
F	R I	M	I	N	U	TT
2	Ha en "ferdig" forbedret bil		Videreutvikle ketegori / medlemskap	Vi sammenligner gruppenes biler og diskuterer hva som er likt/forskjellig. Dette føres videre til samtale om ulike typer biler (spesialisering).	15 min	Ole leder samtalen
				Skrive definisjoner av ting på lapper. Bil: Kjøretøy med 4 hjul og motor… Lastebil: Bil med lasteplan…	15 min	Ole lager passende oppgaver
	Avbrekk og "teambuilding"			LITEN MOROLEK	5 min	Christian
	Ha laget et lite robolab-program	Kjenne Robolab syntaks og GUI.		Kort demo i plenum	10 min	Ole S
				Løse eksempeloppgave Fasit deles ut	15 min	Veiledning av gruppene
	Ha fått bilen til å utføre programmet	Første møte med compilator og overførings-teknologien for program Robolab		Hver gruppe hjelpes med å laste opp programmet de har laget og få bilen til å kjøre. Dette tas etter hvert som de er klare. De raskeste gruppene kan holdes igjen litt ved å initiere diskusjoner om mekaniske eller informatikkfaglige ting eller utfordre dem til å lage det mer avansert.	45 min Vi tar et avbrekk for å spise når det passer	Veiledning av gruppene
				Eksperimentering med bruker-miljøet. Ark med 3-4 ledeoppgaver		Veiledning av gruppene. Roar / Richard lager

				deles ut		ledeoppgaver
F	R I	M	I	N	U	TT
3			Forstå sammen-hengen mellom instruksjoner og utførelse	Kommunikasjonsøvelse. En bak skjermbrett beskriver to tegninger (en abstrakt og en figurativ) for de andre som skal tegne. Diskutere utfallet i gruppen under veiledning. Summere opp i plenum.	15 min	Christian forklarer og leder samtalen
			Kunne omforme instruksjoner til rigid grafisk symbolisme	Lage kort program for å få bilen til å kjøre frem, stoppe og så rygge – tilleggsoppgaver / tips og utg. for diskusjon: Hva må til for å svinge? Hvordan kan man vite at den skal stoppe? Hva skjer hvis det er oppoverbakke? Hovedoppgave og deloppgaver lages på ark som blomst med kjerneoppgave og tilvalgsdeler.	20 min	Veiledning av gruppene
	Avbrekk og "teambuilding"			LITEN MOROLEK	5 min	Christian
	Ha fått bilen til å gjøre noe annet / annerledes	Bli trygge på compilator og overførings-teknologien for program Robolab	Se praktisk effekt av endret instruksjons-sekvens	Gruppene utfordres til å eksperimentere og endre programmet sitt for å se om de kan få bilen til å gjøre andre ting. Her er det viktig at veilederne bidrar med å initiere fruktbare aktiviteter uten å styre/lede for myє. Problematiser syntax ting som forskjell på ikonene for hver av motorene. Hva med løkker og	40 min	Veiledning av gruppene

			forgreninger. Ta tak i ting som oppstår underveis.		
	Rydde opp		Rydde opp	5 min	Alle
	Avslutningslek		LITEN MOROLEK	5 min	Christian

Plan for tirsdag 20.april

1	Løse opp og komme i gang			LITEN MOROLEK	5 min	Christian
	Komme tilbake der vi slapp dagen før	Forsterke og repetere fra dagen før.		Finne frem bilene fra dagen før og sjekke at de fremdeles funker. Bruke litt tid på å få til det siste man ikke rakk på mandag / komme litt videre.	15 min	Veiledning i gruppene
			Møte aggregering som fenomen.	Trekke ut diskusjonen om motorer og syntaks fra gruppesamtalene for å motivere litt samtale om aggregering i plenum. Er det bilen eller motoren som kjører og hva med den andre motoren? Ville de ha kjørt noe sted uten bilen – og omvent?	10 min	Christian leder diskusjonen
		Bli kjent med sensorer	Møte aggregering som fenomen.	Samtalen videreføres mot sensorer som sikkert noen av gruppene har diskutert uten å bruke det enda	5 min	
		Bli kjent med sensorer		Introdusere sensorer. Kort demo	10 min	Richard tegner og forteller
		Prøve å bruke sensorer selv og derigjennom videreutvikle ferdigheter i Robolab	Stadig forbedret kjennskap til prinsipper og virkemåte for imperativ programmering	Gruppene får prøve seg selv. Ark med blomst deles ut. Kjerneoppgave er veldefinert, mens tilvalgsoppgavene kan være mer på hint nivå. De siste følges opp av veilederne.	30 min	Veiledning av gruppene
F	R I	M	I	N	U	TT
2	Problem-definisjon			Starte med innledende diskusjon i plenum om ulike roboter i	10 min	Richard leder samtalen

				hverdagen. Brainstorm-aktig liste på tavlen etter hvert som det kommer forslag. Aktiviteten må "styres" en del for å pense inn på varianter som de evt kan klare å lage selv.		
				Gruppene setter seg sammen for å velge problemstilling. De skal både bestemme hva de vil lage og hva de evt må finne svar på for å få det til eller som de vil finne ut etter at de er ferdige.	30 min	Veiledning
	Avbrekk og "teambuilding"			LITEN MOROLEK	5 min	Christian
	Problemløsing del 1			Gruppene jobber med hver sine oppgaver og veiledes underveis. Viktig at veileder stiller "lure" spørsmål og hjelper elevene inn på faglig spennende diskusjoner og utforskinger	60 min med avbrekk for eller avsluttet av spising	Veiledning
F	R I	M	I	N	U	TT
3			Klassifisering og spesialisering gjenspeilt i syntaks / formalspråk.	Diskusjon om motorer og sensorer som instanser av kategorier og om ulike typer sensorer og deres spesielle egenskaper. Hvordan gjenspeiles dette gjennom syntaks i Robolab. Hva er likt? Hva er forskjellig		Christian
	Avbrekk og			LITEN MOROLEK	5 min	Christian

	"teambuilding"					
	Problemløsing del 2			Gruppene jobber med hver sine oppgaver og veiledes underveis. Viktig at veileder stiller "lure" spørsmål og hjelper elevene inn på faglig spennende diskusjoner og utforskinger		Veiledning

Plan for onsdag 21.april

1	Problemløsing del 3			Gruppene jobber med hver sine oppgaver og veiledes underveis. Viktig at veileder stiller "lure" spørsmål og hjelper elevene inn på faglig spennende diskusjoner og utforskinger		Veiledning
	Ekstraoppg. ?			Kameraoppgave. Gruppene skal få roboten til å kjøre inn bak en vegg for å filme det som er der.		Richard
F	R I	M	I	N	U	TT
2	Ekstraoppgave forts. ?			Kamerautstyret går på omgang mellom gruppene som får prøvet sitt program etter hvert.		Veiledning
	Problemløsing del 4			Gruppene jobber med hver sine oppgaver og veiledes underveis. Viktig at veileder stiller "lure" spørsmål og hjelper elevene inn på faglig spennende diskusjoner og utforskinger		Veiledning
F	R I	M	I	N	U	TT
3	Problemløsing del 5			Gruppene jobber med hver sine oppgaver og veiledes underveis. Viktig at veileder stiller "lure" spørsmål og hjelper elevene inn på faglig spennende diskusjoner og utforskinger		Veiledning
				Oppsummering og repetisjon	30 min	

Plan for torsdag 6.mai

	Lage et klassehierarki for dyr		Repetisjon av klassifisering og spesialisering	Tegne hver sin tegning av et dyr. Tegningene skal henges opp på et stort tre i grupper og undergrupper.	15 min	Richard introduserer og instruerer
				Diskutere egenskaper, slektskap, forskjeller etc. Lage et tre for organisering av levende vesener.	10 min	Christian leder diskusjonen

Dag 1:			
Bygge hus	ind	Ole	20 min
Diskutere hus	Gruppe/plenum	Ole	
Rydde klossene	Gruppe / plenum	Ole	10 min
Bygge bil	gruppe	veiledning	50 min
PAUSE			
Sammenligne biler	plenum	Ole	15 min
Skrive definisjoner	Gruppe / plenum	Ole	15 min
Demo Robolab	plenum	Ole	10 min
Prøve Robolab	grupper	veiledning	45 min (inkl mat)
PAUSE			
Beskrive tegning	grupper	Christian	15 min
Fortsette Robolab	grupper	veiledning	60 min
Rydde ++	alle		
Dag 2:			
Finne frem igjen bilene	gr	veil	15 min
Diskusjon om aggregering / introdusere sensorer	plenum	Christian	15 min
Demo sensorer	Plenum	Richard	10 min
Prøve sensor	grupper	veil	30 min inkl mat
PAUSE			
Brainstorm	plenum	Richard	10 min
Problemdefinisjon	grupper	veil	30 min
Problemløsing 1	grupper	veil	60 min (inkl mat)
PAUSE			
Diskusjon om subklasser og metodekall	Plenum	Christian	30 min
Problemløsing 2	grupper	veil	60 min
Dag 3:			
Problemløsing 3	grupper	veil	
Problemløsing 4	grupper	veil	
Problemløsing 5	grupper	veil	

1

```
// Disse to linjene forteller programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;
```

2

```
// Her starter programmet vårt.
class <NAVNET PÅ PROGRAMMET> {
    public static void main(String [] args) {
        new Kontroll();
    }
}
```

●

```
class Kontroll {
      // først sier vi hva programmet skal bestå av.
      // I dette eksempelet skal vi ha en robot som heter volvo
      // og en assistent som heter per
      Robot volvo;
      Assistent per;

      Kontroll(){

            //her lager vi en ny bil som vi skal programmere:
            volvo = new Robot();
            //og så lager vi en ny assistent:
            per = new Assistent();

            // *******************************************
            //her kan dere skrive hva roboten skal gjøre

            // *******************************************
      }

}
```

~~//nå gjenstår det å lage selveste bilen~~

●

4

```
class Robot{
      //først må vi si hva roboten består av
      Legomotor venstreMotor;
      Legomotor høyreMotor;
      //osv.....

      Robot(){
            // Helpevariabler for motorer. Ikke bry dere om disse
            int A = 1; int B = 2; int C = 3;

            //her lager vi alt som roboten skal bestå av

            //en motor festet til port A på RCXen
            venstreMotor = new Legomotor(A);
            //og så en motor festet til port C på RCXen
            høyreMotor = new Legomotor(C);
            //osv.....

      }

}
```

```
// ********************************************
```

class Legomotor

```
// metoder:
```

> **kjørFremover(kraft)**
> Metode som sier at motoren skal kjøre fremover med en kraft fra 0 til og med 7.
>
> **kjørBakover(kraft)**
> Metode som sier at motoren skal kjøre bakover med en kraft fra 0 til og med 7.
>
> **stopp()**
> Metode som stopper en motor (bråstopp)
>
> **mykStopp()**
> Metode som slår av strømmen til en motor (ikke bråstopp)

```
// ********************************************
```

class Assistent

```
// metoder:
```

> **vent(millisekunder)**
> Metode som får programmet til å vente et gitt antall millisekunder før programmet fortsetter.
>
> **tilfeldigTall(makstall)**
> Metode som gir et tilfeldig tall mellom 0 og det medfølgende tallet.

```
// ********************************************

class Trykksensor extends Sensor

// metoder:

    trykkesInn()
    Metode for å spørre trykksensoren om den er trykket
    inn. Svarer den true, så er den trykket inn.


// ********************************************

class Lyssensor extends Sensor

// metoder:

    lysstyrke()
    Metode for å få vite lysstyrken som lyssensoren ser
    akkurat nå. Dette er en Verdi mellom 1 og 100.


// ********************************************

class Rotasjonssensor extends Sensor

// metoder:

    rotasjon()
    Metode for å få vite hvor mange ganger
    rotasjonssensoren har rotert.

    reset()
    Metode for å sette rotasjonssensoren til 0.
```

```
// *********************************************
```

class Speaker

```
// metoder:
```

spillTone(frekvens, lengde)
Metode som spiller en tone på angitt frekvens.
Hørbare frekvenser er mellom 21 og 2100 Hz.
Lengden angis i hundredels sekunder.

```
// *********************************************
```

class Display

```
// metoder:
```

skriv(tall/tekst)
Metode som skriver et tall (maks 5 siffer) til
displayet. Hvis en tekst skal skrives til displayet,
så må teksten stå i gåseøyne ("molle"). Maks 5 tegn.

******************** OPPGAVE *********************

Dere skal lage et program som simulerer aktiviteten i en resturant.
Mao – en resturantrobot... ☺

Denne resturanten har en kokk, en hovmester, en meny og mange bord.
All info dere trenger for å løse oppgavene er kort forklart på de neste sidene.
Bruk derfor disse arkene masse hele tiden for å sjekke hva som finnes.

Her er oppgaven:
Det kommer fire gjester (som dere bestemmer hva heter).
Disse skal ha et bord og spise en hyggelig middag med dessert.
Dere skal bestemme hva de vil spise fra rettene på menyen.

Problem 1:
Når de kommer inn er det kun ett bord ledig: bord4.
Det er kun 2 stoler/plasser ved dette bordet. Men de er jo 4...

Når de har kommet på plass ved bordet, skal de velge hva de vil spise fra menyen,
kelneren skal ta imot Bestilling som leveres til kokken, og kokken skal lage maten

Problem 2:
Maten blir ferdig fra kjøkkenet i hulter til bulter rekkefølge.
Servitøren må derfor finne ut hvem som skal ha hva slags mat for å levere riktig
hovedrett og dessert til hver av de fire gjestene.

P.S. Husk at de må spise hovedrett før dessert.

Skriv et program som gjør at alt beskrevet i oppgaven over skjer. Dersom
dere har gjort alt riktig vil det komme ut en testutskrift på skjermen
som forteller dere at alt er riktig.

***************** De første to klassene er synlige i deres program *************
*********************og dere kan lage endringer på dem ******************

class Kontroll

Det er her dere kontrollerer hva som skal skje i programmet.
Her må dere lage et Resturant-objekt. F.eks:

```
    Resturant vertshuset;
    vertshuset = new Resturant;
```

I tillegg må vi ha noen Personobjekter som skal bli gjestene etter hvert.
Etter at det er gjort, kan dere begynne å gi kommandoer til de ulike personene i resturanten slik at alle til slutt får det de har bestilt (forhåpentligvis).

#--

class Resturant

```
Kelner hovmester;
Kokk tor;
Meny matliste;
Bord bord1;
Bord bord2… osv
```

Det finnes flere kelnere i resturanten, men disse tilhører bordene.

#--

********************** De neste klassene har vi laget **********************
****************** Her skal dere finne info om hva de inneholder *************
************** og hvilke ting dere kan be objektene av dem om å gjøre. *********

class Person("<navn>", "<kjønn>")

Når Personer må få navn og kjønn oppgitt, så betyr det at også Gjester, Kelnere og Kokker må få det, siden de er spesialtilfeller av Personer.

#--

class Gjest **extends Person**

Representerer en gjest på resturanten.

`velgFraMeny(<hovedrettNr>, <dessertNr>)`
Velger hovedrett og dessert fra menyen. Disse valgene er angitt som tall, der hvert tall representerer en matrett på menyen.

`hovedrett()`
Sender tilbake nummeret på hovedretten som denne gjesten har bestilt.

`dessert()`
Sender tilbake nummeret på desserten som denne gjesten har bestilt.

#--

class Kokk **extends Person**

Representerer en kokk på resturanten.

`lageMat():`
Lager maten han har fått på bestilling.

#--

class Kelner extends Person

Representerer en kelner i en resturant. Hver kelner tilhører ett eller flere Bord, dersom de ikke er hovmester.

`endreAntallPlasser(<bord>, <antall>)`
Dette er det normalt hovmesteren som gjør, men andre Kelnere kan også gjøre det. Endrer det antallet stoler som står ved et bord til det antallet som er gitt. OBS: De som allerede sitter ved bordet forsvinner da!

`plasserVedBord(<gjest>, <bord>)`
Plasserer en gjest ved et bord. Dette er det hovmesteren som gjør.

`taBestilling(<gjest>)`
Tar en bestilling fra en gjest, tar både hovedrett og dessert samtidig, og leverer bestillingen til Kokken tor.

`hentHovedrett()`
Henter en hovedrett fra kjøkkenet. Kelneren kan bare bære ett fat med mat av gangen.

`hentDessert()`
Henter en dessert fra kjøkkenet. Kelneren kan bare bære ett fat med mat av gangen.

`finnRiktigGjest(<bord>)`
Når kelnerern har hentet en matrett og skal tilbake til bordet med denne, spør metoden hvem som har bestilt matretten kelneren kommer med. Metoden gir et svar som er av typen Person og svaret peker på gjesten som hadde bestilt den matretten kelneren kom med.

`leverMatrett(<gjest>)`
Leverer matretten kelneren har hentet fra kjøkkenet til den gjesten som er angitt i parentesen.

#---

class Bord

```
Kelner servitør;
Gjest gjest1;
Gjest gjest2;
Gjest gjest3... osv
```

avh av hvor mange som har blitt plassert ved bordet av hovmesteren.

#--

class Meny

```
Hovedretter:
"Biff", "Elgstek", "Salat", "Laks", "Kotelett", "Oksestek"
Desserter:
"Is", "Kake", "Bær", "Sjokoladepudding", "Gele", "Tiramisu"
```

`rettNummer("<matrett>")`
Metoden sjekker navnet på matretten, og sier ifra hvilket nummer retten har.

#--

class Display

`skriv("<teksten dere vil skrive ut>")`
Denne kan dere bruke til å lage testutskrifter underveis for eksempel for å vise dere selv hvor langt programmet har kommet.

#--

En liten instruksjonsmanual for klargjøring av windowsmaskiner til å bruke Lejos

Roar Granerud
Institutt for Informatikk
Universitetet i Oslo
E-post: rgraneru@ifi.uio.no

16. mars 2004

Hva som trengs av programvare på maskinene

- Java System Development.exe kit må legges inn. Dette er en fil ved navn j2sdk-1_4_2_03-windows-i586-p.exe som må kjøres. Følg så instruksjonen på skjermen i forbindelse med installasjonen. Husk hvor du legger programvaren. bin-katalogen i SDKen må legges inn i pathen for at windows skal vite hvor javac.exe befinner seg. Det er mulig dette gjøres automatisk. Manuelt gjøres det ved i System Properties, Advanced, Environment Variables.
- Lejoskatalogen kopieres så over til et ønsket sted på harddisken. Husk også på hvor denne legges.
- RCXDownload kopieres så over til et ønsket sted på harddisken. Dette er et grafisk brukergrensesnitt for compilering, linking og overføring av lejos-kode til RCXen. Startfilen til RCXDownload heter RCXDownload.bat. Det er ingenting som skal installeres i denne sammenheng sålenge Java SDK er lagt inn.
- En gratis og enkel IDE ved navn Jcreator3 legges inn på maskinen. Start setup.exe og følg instruksjonene på skjermen. For at denne IDEen skal se litt enklere bør den ene raden med knapper fjernes ifra GUIen. Den som kan fjernes heter Tools og ligger på view, Toolsbars.

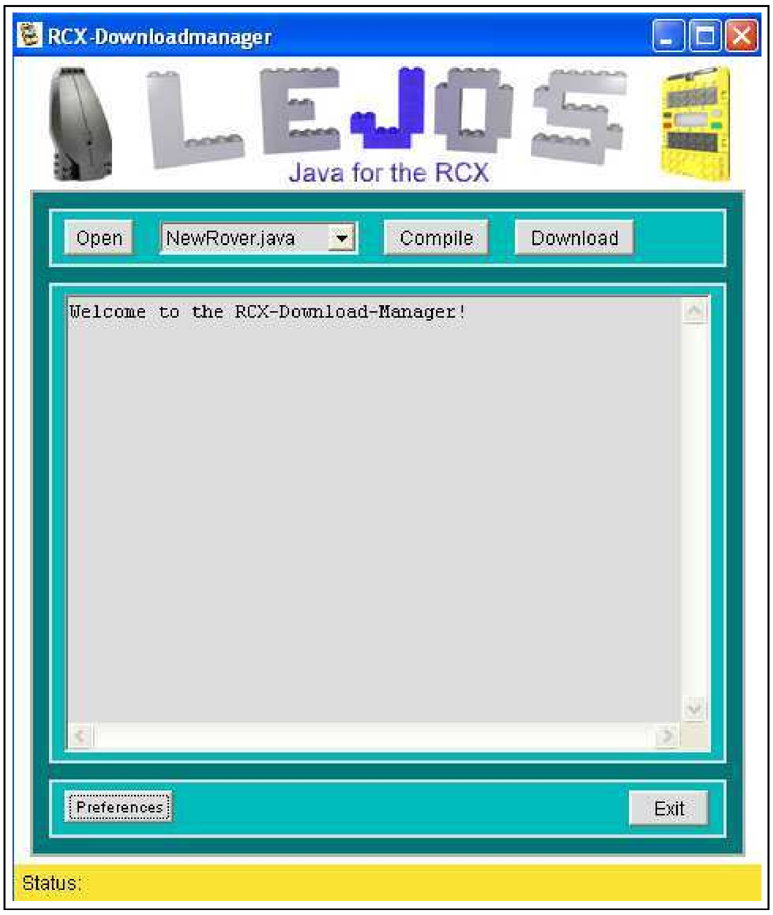
Bruk av RCXDownload

RCXDownload er en liten GUI laget i Java for kompilering, linking og overføring av Lejosprogrammer fra datamaskinen til RCXen. GUIen har 4 vesentlige knapper. Preferences knappen må brukes idet programmet er lagt over på maskinen det skal brukes på. Her kan du vise RCXDownload hvor Lejos og Java SDK er installert. Det må også bestemmes hvilken serieport som skal brukes til overføringen. Dette må gjøres før RCXDownload kan brukes. De andre valgene i Preferences er ikke vesentlige. Den andre knappen er open og brukes for å hente hvilken javafil som skal kompileres og overføres. Dette er da filen med main-metoden, men vi kommer vel ikke til å dele opp programmene i flere filer uansett. Den tredje knappen er compile. Den kompilerer Lejosprogrammet til .class-filer. Feilmeldinger blir skrevet ut på skjermen i RCXDownload. Den siste knappen, Download, kompilerer og linker .class-filene til en binærfil som så overføres til RCXen. Pass på at RCXen står i overføringsmodus (Slå den av og på hvis den er ugrei) og pass på at lyset i rommetikke ødelegger for overføringen.

Merk at denne versjonen av RCXDownload er modifisert av meg slik at bl.a knappen for overføring av Lejos-Firmware mangler. Dette for at elevene og vi ikke skal trykke på denne ved en feiltagelse. Jeg regner med dette ikke blir noe problem siden vi skal ha med ferdig klargjorte RCXer.

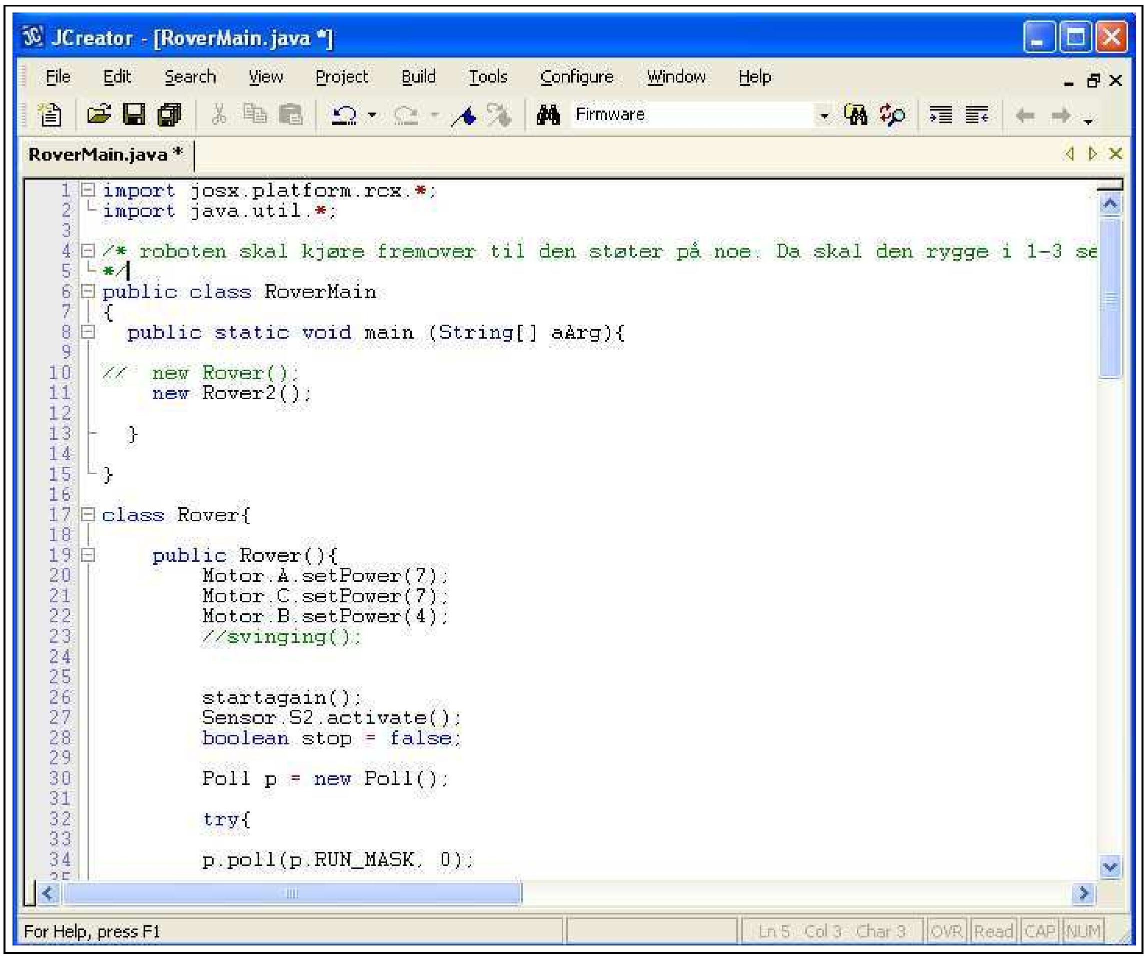
Bruk av JCreator

Dette programmet er en standard teksteditor for windows. En litt mere avansert versjon av Notepad. Den har iallefall fargekoder på javaprogrammeringen. Det eneste den skal brukes til er å lagre .java-filene. RCXDownload tar seg av resten. Hvis noen vil prøvde den er den tilgjengelig for nedlasting ifra: http://www.jcreator.com/download.htm



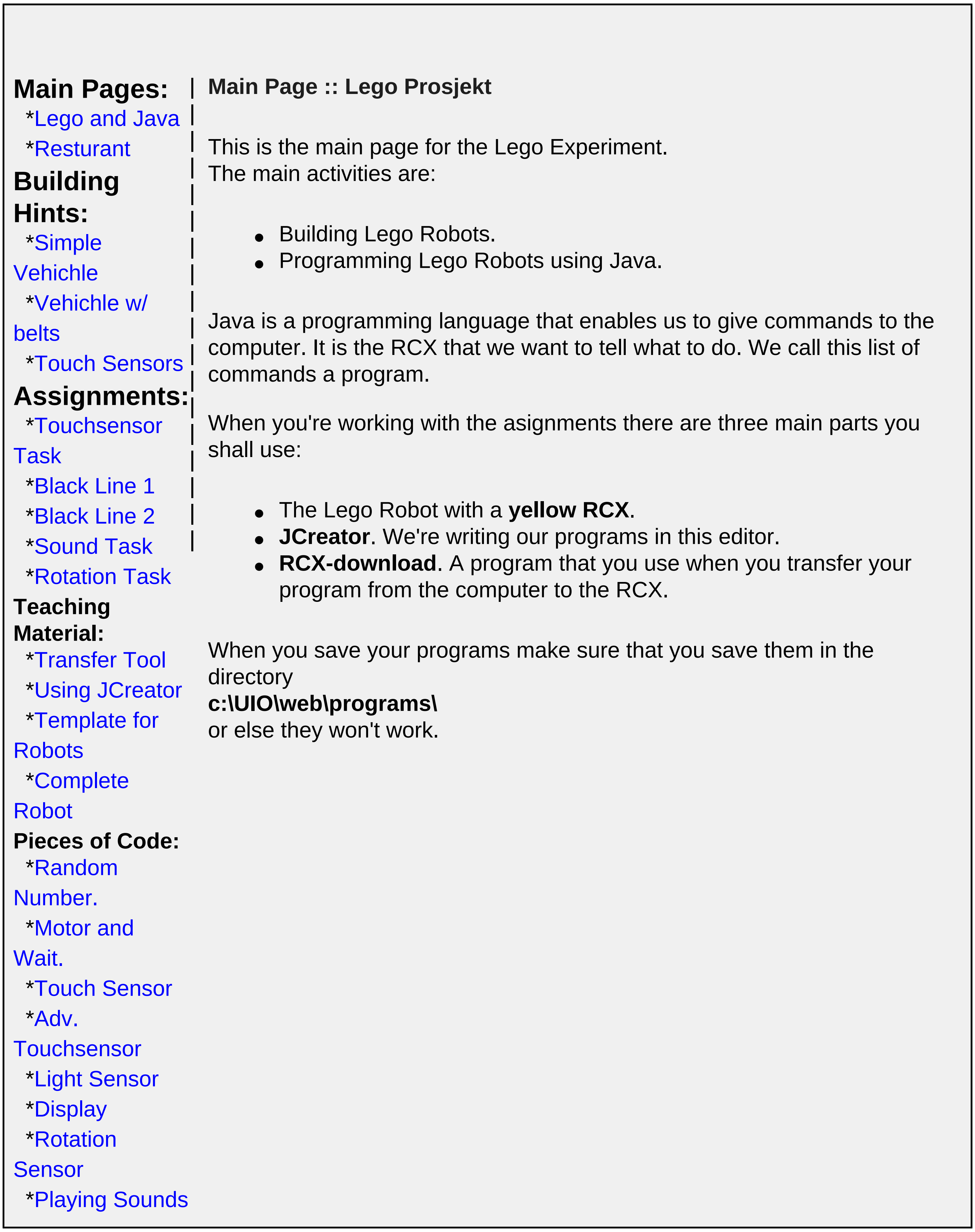
```
Welcome to the RCX-Download-Manager!
```

Figur 1: RCXDownload GUI


JCreator - [RoverMain.java *]
File Edit Search View Project Build Tools Configure Window Help
Firmware
RoverMain.java *
For Help, press F1
Ln 5 Col 3 Char 3
OVR Read CAP NUM

```
import josx.platform.rcx.*;
import java.util.*;

/* roboten skal kjøre fremover til den støter på noe. Da skal den rygge i 1-3 se
*/
public class RoverMain
{
  public static void main (String[] aArg){

//  new Rover();
    new Rover2();

  }

}

class Rover{

    public Rover(){
        Motor.A.setPower(7);
        Motor.C.setPower(7);
        Motor.B.setPower(4);
        //svinging();


        startagain();
        Sensor.S2.activate();
        boolean stop = false;

        Poll p = new Poll();

        try{

        p.poll(p.RUN_MASK, 0);
```

Figur 2: JCreator teksteditor



Main Pages:
*Lego and Java
*Resturant
Building Hints:
*Simple Vehichle
*Vehichle w/ belts
*Touch Sensors
Assignments:
*Touchsensor Task
*Black Line 1
*Black Line 2
*Sound Task
*Rotation Task
Teaching Material:
*Transfer Tool
*Using JCreator
*Template for Robots
*Complete Robot
Pieces of Code:
*Random Number.
*Motor and Wait.
*Touch Sensor
*Adv. Touchsensor
*Light Sensor
*Display
*Rotation Sensor
*Playing Sounds

Main Page :: Lego Prosjekt

This is the main page for the Lego Experiment.
The main activities are:

- Building Lego Robots.
- Programming Lego Robots using Java.

Java is a programming language that enables us to give commands to the computer. It is the RCX that we want to tell what to do. We call this list of commands a program.

When you're working with the asignments there are three main parts you shall use:

- The Lego Robot with a **yellow RCX**.
- **JCreator**. We're writing our programs in this editor.
- **RCX-download**. A program that you use when you transfer your program from the computer to the RCX.

When you save your programs make sure that you save them in the directory
c:\UIO\web\programs\
or else they won't work.

Resturantoppgaven

Vi har en resturant som består av personer, kelnere, kokker, bord osv. Det er deres jobb å sørge for at de forskjellige personene sitter ved bord, at bestillingene deres blir tatt og at kokken lager riktig mat til riktige personer.

Under er en link til en startfil. Denne åpnes i Jcreator på vanlig måte når dere trykker på den. Det står hvor i denne filen dere skal skrive deres egen del av programmet.

Når dere er fornøyd med programmet så langt lagrer dere det på vanlig måte (save). Deretter må programmet kompileres. Dette ble før gjort i RCXTools. Nå må dere gjøre det manuelt med et kommandovindu. Dette vinduet får dere frem ved å trykke på start-knappen nederst til venstre, velge kjør, og så skrive cmd (etterfulgt av enter).

Dere vil nå ha fått opp et svart vindu med en kursor blinkende. Dere må nå gå til den riktige katalogen. Det gjør dere med kommandoen:

`cd   \uio\resturant` (etterfulgt av enter)

Det vil nå stå `C:\UIO\Resturant>` nederst i det svarte vinduet.

For å kompilere programmet skriver dere:

`javac MollaMat.java` (etterfulgt av enter....pass på store bostaver og sånt)

Hvis dere ikke får noen feilmeldinger skriver dere følgende:

`java MollaMat` (etterfulgt av enter)

Da vil programmet deres kjøre. Lykke til...det vil dere trenge...og en ting til. Pass på å lage programmet i små porsjoner og kompiler ofte, slik at dere ikke får så mange feilmeldinger om gangen.

Resturantoppgaven MollaMat

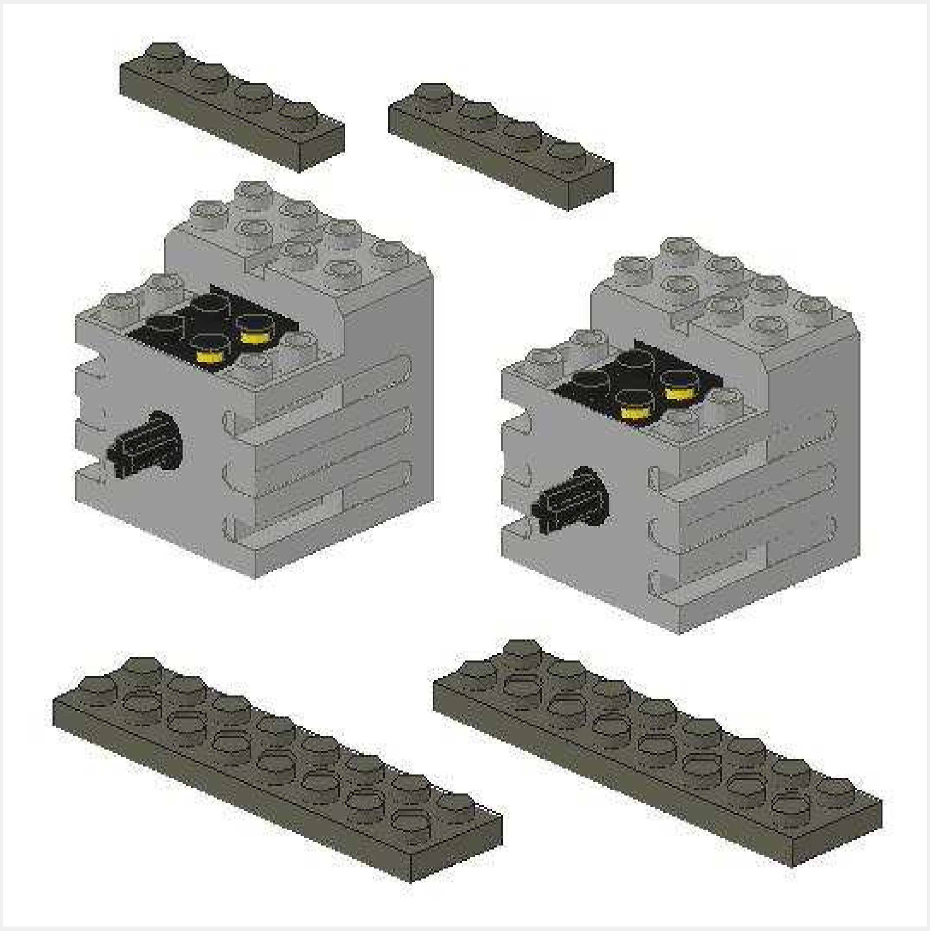
Main Pages:
*Lego and Java
*Resturant
Building Hints:
*Simple Vehichle
*Vehichle w/ belts
*Touch Sensors
Assignments:
*Touchsensor Task
*Black Line 1
*Black Line 2
*Sound Task
*Rotation Task
Teaching Material:
*Transfer Tool
*Using JCreator
*Template for Robots
*Complete Robot
Pieces of Code:
*Random Number.
*Motor and Wait.
*Touch Sensor
*Adv. Touchsensor
*Light Sensor
*Display
*Rotation Sensor
*Playing Sounds

Building Hints :: Simple Vehicle

The simple vehicle is the three first steps on the vehicle with belts, but instead is has 2 wheels and little else.

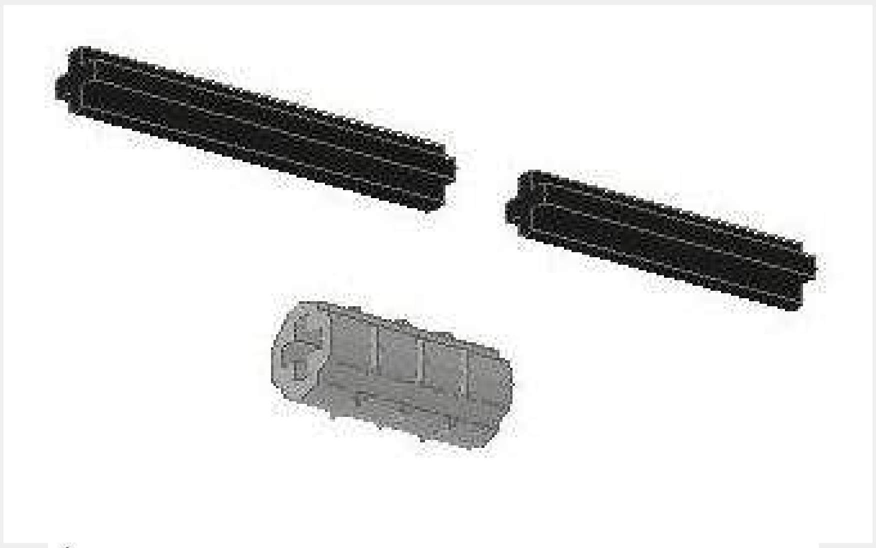
Building Hints :: Vehicle with belts

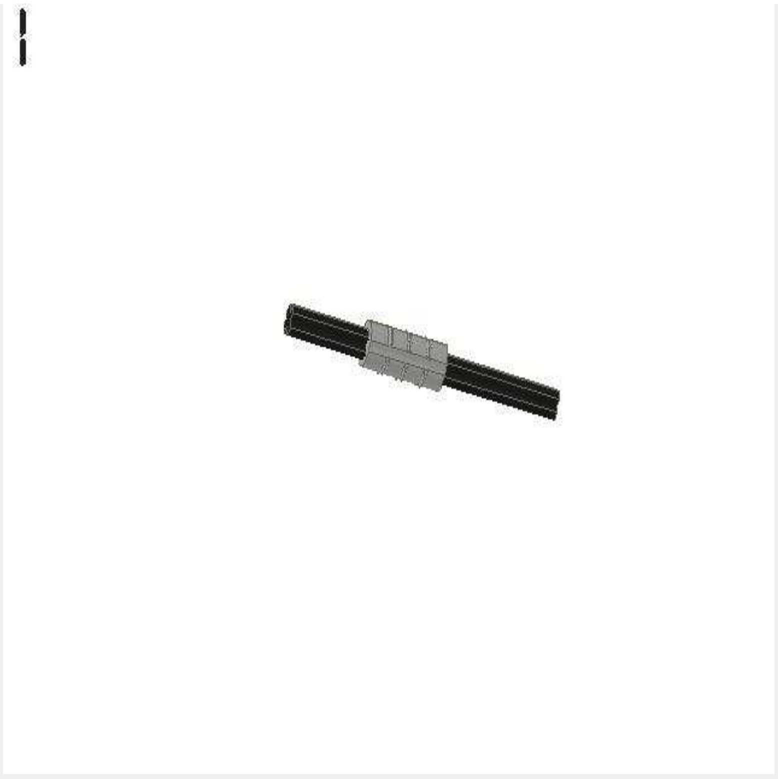
The best thing about the vehicle with belts is that it's very stable and it is easy to turn. The picture on the left shows the vehicle from below and the picture on the right shows the vehicle from above. The topmost picture is just a picture of the parts you need for the step. Don't mind that the colors on the pieces in the pictures. Some of them are of a different color so that you will find it easier to tell them apart.



Building Hints :: A bumper with two touch sensors

A bumber of this kind is a good thing to make the robot understand that it has hit an object in its path, and needs to find a different route. I don't think that you will have any problems putting it on your robot.





Assignment :: Touch Sensors

Make a robot that drives forward until it hits an obstacle. When it does, make it back off a little, turn a little to the left and then stop.

This robot needs only one touch sensor.

When you have completed the first part you can make a smarter and more advanced robot. A robot that can drive around on the floor, and when it hits something, it backs off and tries go clear of the object it crashed into.

To make this kind of robot you need to use 2 touch sensors. I recommend that you build the kind of bumber found at the building hints. Just attach this bumper to the car and attach the wires to the correct ports.

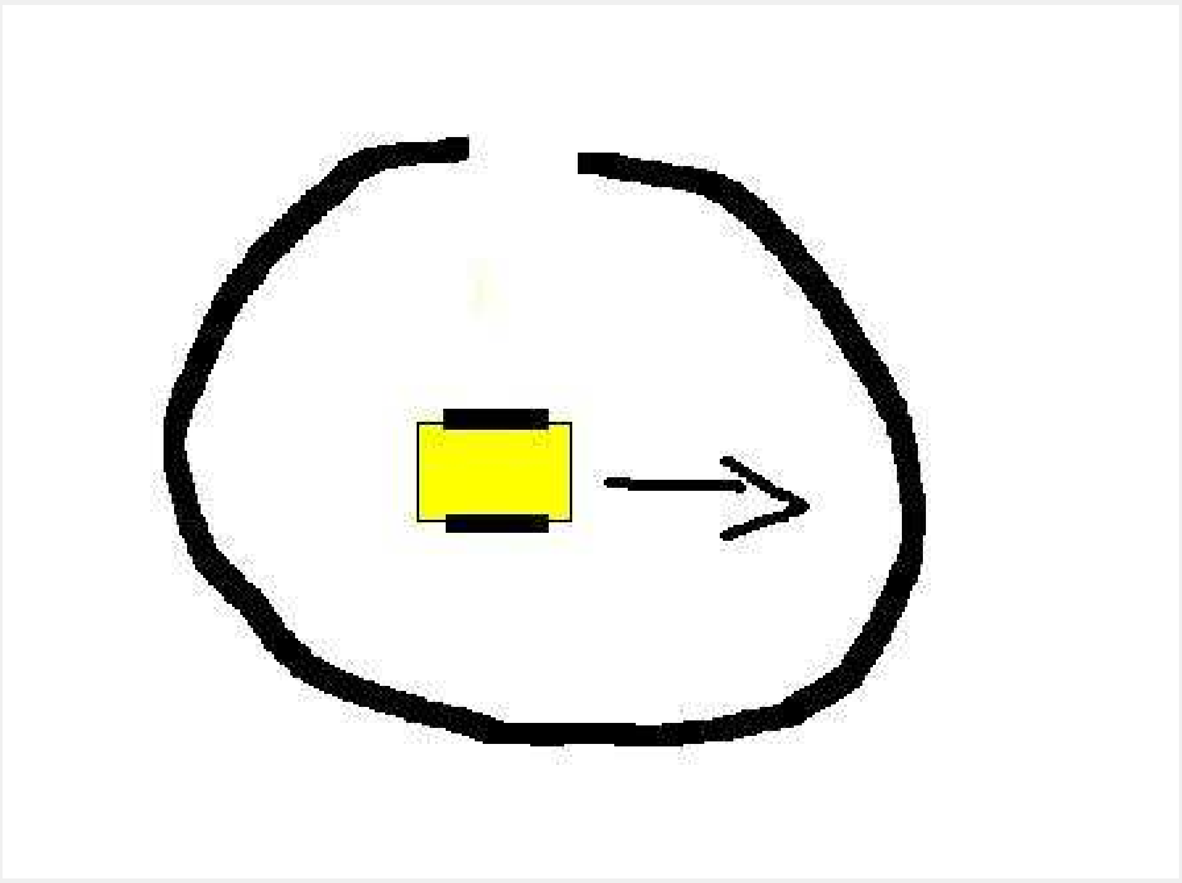
The robot you're making now shall drive forward until one of the touch sensors on the bumber is activated. That the button on the touch sensor is pushed. Then I want the robot to back off a little distance(perhaps a random distance between 0 and 2 seconds) and then turn left or right, depending on which touch sensors that was pushed. And then just make the robot continue forward until it hits something else.
This robot shall run forever, and to make it do just that you need to write your program inside a while loop that looks like this:

```
while(true) {
   write your program here
}
```

Write the program one part at a time so it will be easier to understand. Good luck, you will need it

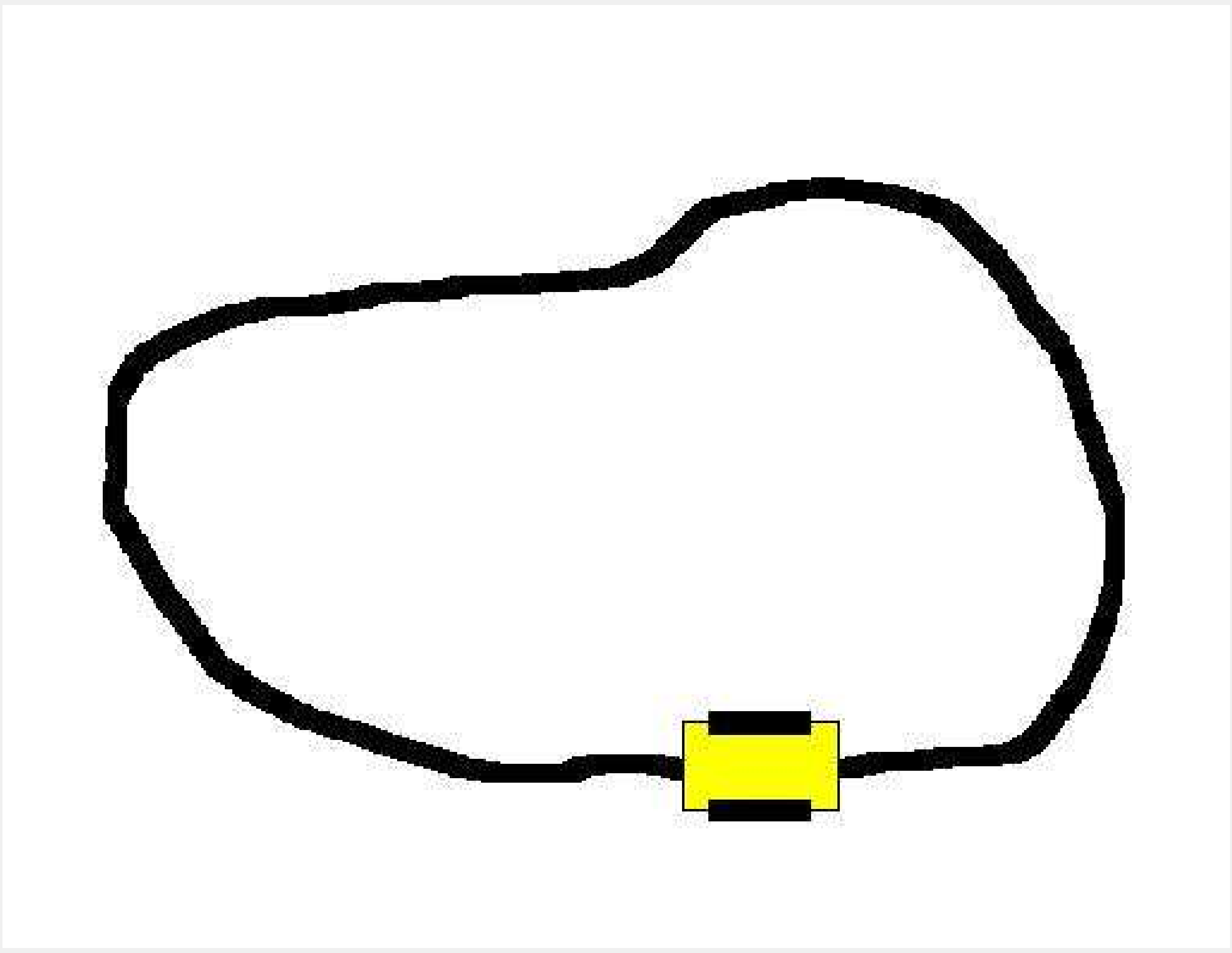
Assignment Lightsensor 1 :: Find the way out of a black circle

Place your robot inside a black sirkel that has a little opening. Using a light sensor your robot shall be able to find the opening of the circle, and not run over the black border. You are not allowed to place the robot in front of the opening. :)



Question: How do you know how dark the circle is?

Assignment Lightsensor 2 :: Follow the black tape



In this assignment you have to make your robot follow the black tape. To be able to follow the tape you have to use light sensors. How many light sensors do you need to be able to follow the black tape? How shall the light sensors behave when they see the black tape? You also have to consider how fast your robot can move in order to notice the black tape.

Assignment :: Use the RCX to play a song

Create a program that plays a little piece of music. To do this you must connect a Speaker to your robot to be able to use it to play a tone. Use playNote(frequency, length) where the frequency is a number below 2100Hz for the human ear to be able to hear the tone. The length is measured in 1/100 of a second. If you type playTone(500,100) it will play a pretty deep tone in one second (100/100 seconds)

To be able to solve this assignment you have to find out what frequency the different tones is. For example you cound find that a frequency of 700 is the same a C tone.

To make the assignment easier and the program less messy you should make a method for each of the tones. So when you want to play 3 tones after eachother you only have to write:

```
playA();
playC();
playLongA();
```

and so on.

You have to make these methods yourself.

And again....Good Luck

Assignment :: Rotation Sensor

To solve this assignment you have to place a rotation sensor on one of the axles on the robot, and make sure that the axle turns around so that the movement is registered. When you connect the rotation sensor the robot may turn a little to the left or the right when it is supposed to go forward. You have to compensate for this skewness by making the motors go with differend power.

In this assignment I want you to make the vehicle go forward for about a meter. And then it shall play a sound. Then is shall turn 180 degrees and go 1 meter the other way so that it is in the same place as it started..

But on the way back I want to make the robot go a little slower. So to make it go one meter the other way you have to know how far it has gone, so you can find out when to stop. Use the rotation sensor for this task. You can ask the rotation sensor how many rotations it has recorded and then ask it when this number of rotations occur again.

It is very important that you solve this assignment in small steps at a time, so that it will be easier to understand

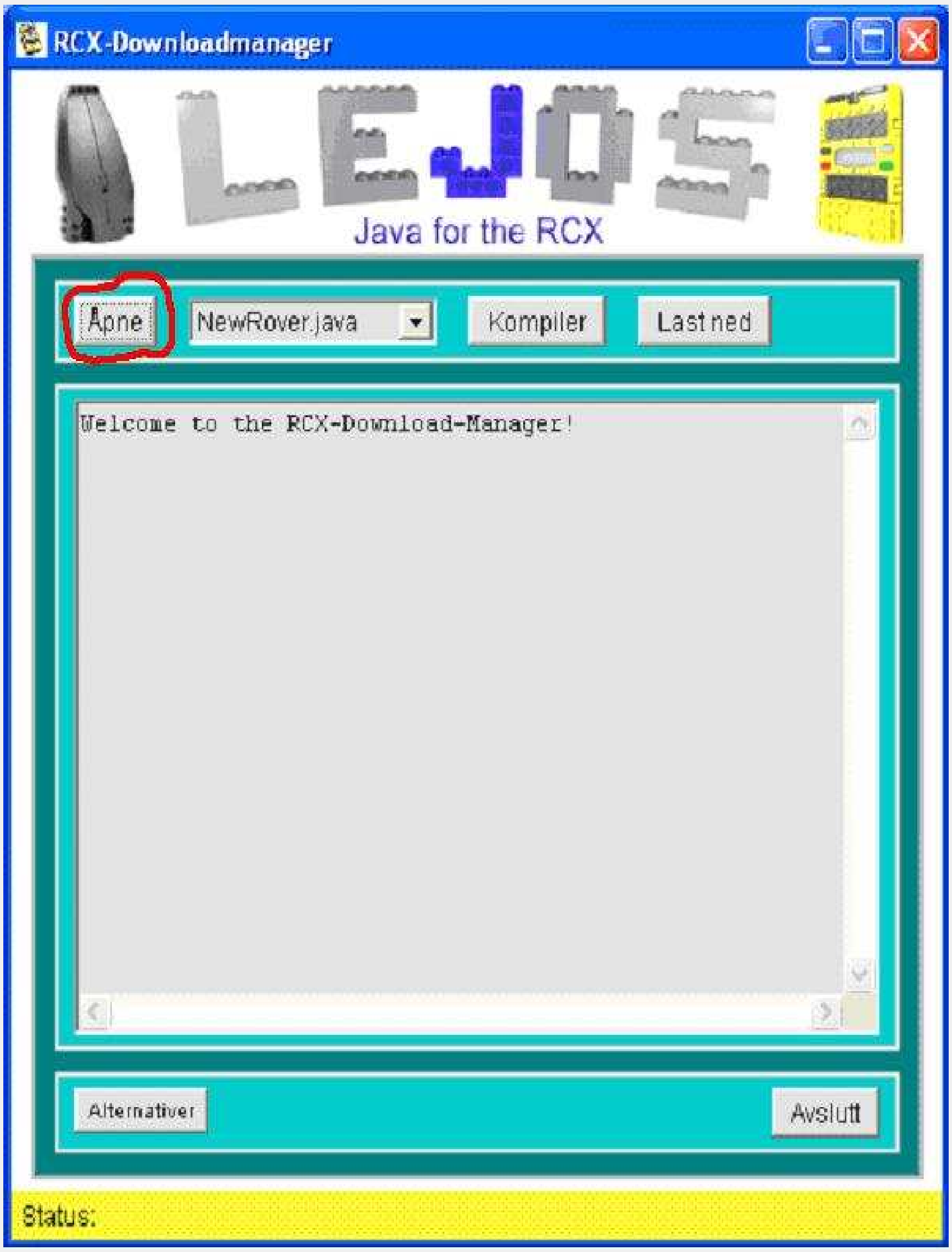
It may be difficult to make the robot run straight forward when you have connected the rotation sensor. This is not a big deal. As long as you record the distance the robot has travelled (approximately)

Good Luck.

Teaching Material :: How to use RCXDownload

To make the robot do our bidding we have to transfer the program we have written from the computer to the RCX. To do this you need a program called RCX-download. This is a very easy program to use

To start the program, double-click the RCX-download icon on the desktop. Then a window will appear:

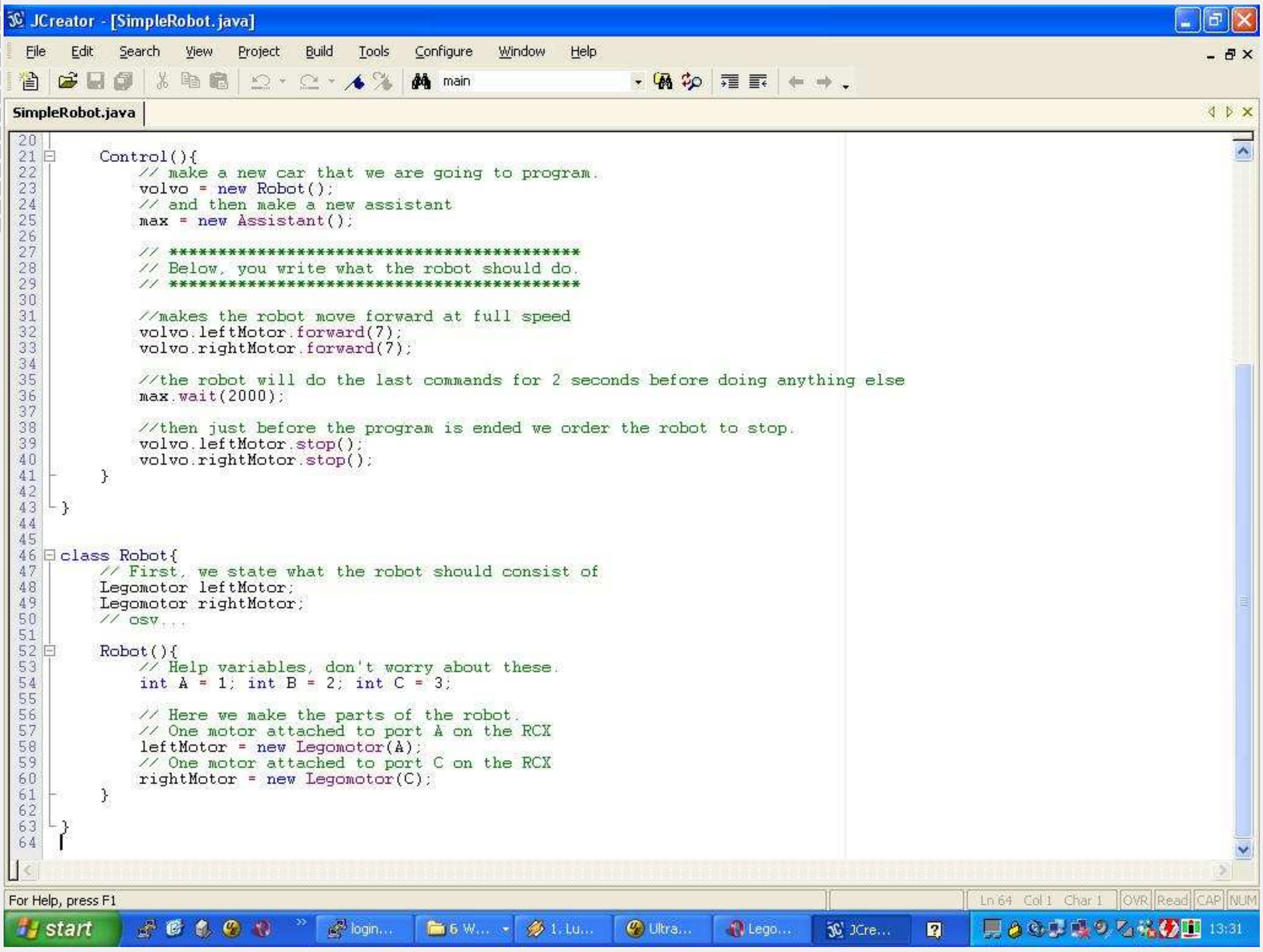


```
Welcome to the RCX-Download-Manager!
```

This is our main window. We can se that there is a button called "Open" here. Press this button to select a new java-program to transfer to the RCX. When you press "Open" the following window pops up:

How to use the editor JCreator

When we're writing our programs for the robot we have to use some kind of editor. We've chosen an editor called JCreator. The shortcut the the program is on your desktop. Lets look at a picture of the program:


JCreator - [SimpleRobot.java]
File Edit Search View Project Build Tools Configure Window Help
main
SimpleRobot.java
For Help, press F1
Ln 64 Col 1 Char 1 OVR Read CAP NUM
start
13:31

```
    Control(){
        // make a new car that we are going to program.
        volvo = new Robot();
        // and then make a new assistant
        max = new Assistant();

        // *******************************************
        // Below, you write what the robot should do.
        // *******************************************

        //makes the robot move forward at full speed
        volvo.leftMotor.forward(7);
        volvo.rightMotor.forward(7);

        //the robot will do the last commands for 2 seconds before doing anything else
        max.wait(2000);

        //then just before the program is ended we order the robot to stop.
        volvo.leftMotor.stop();
        volvo.rightMotor.stop();
    }

}


class Robot{
    // First, we state what the robot should consist of
    Legomotor leftMotor;
    Legomotor rightMotor;
    // osv...

    Robot(){
        // Help variables, don't worry about these.
        int A = 1; int B = 2; int C = 3;

        // Here we make the parts of the robot.
        // One motor attached to port A on the RCX
        leftMotor = new Legomotor(A);
        // One motor attached to port C on the RCX
        rightMotor = new Legomotor(C);
    }

}
```

Don't mind all the text in the editor. You can notice a lot of similarities with for example Microsoft Word: We can save and open files, cut, paste and so on. You also notice all the different colors on the words in the file. Blue, green, brown, purple and black. These different colors means different things:

- Blue: These are reserved words in Java. Reserved means that these words has a spesial meaning in Java, and these words cannot be used for other things than what they're reserved for.
- Green: Everything that is written in green is a commentary. It is never read by the RCX. It is just small reminders to the programmer to easier understand the program later on.
- Purple: When you give a method a name, and when you use a method the color is purple. A method is a command that does spesific things. You can also make your own methods. turnLeft is an example of a method.
- Black: This is the rest of the program which isn't methods, reserved or anything special

A little hint: When you save your programs, save it in the same folder each time. First of all it will be easier for you to find it when you need it, and there are some extra stuff in the folder that the program needs to work.

Main Pages:
*Lego and Java
*Resturant
Building Hints:
*Simple Vehichle
*Vehichle w/ belts
*Touch Sensors
Assignments:
*Touchsensor Task
*Black Line 1
*Black Line 2
*Sound Task
*Rotation Task
Teaching Material:
*Transfer Tool
*Using JCreator
*Template for Robots
*Complete Robot
Pieces of Code:
*Random Number.
*Motor and Wait.
*Touch Sensor
*Adv. Touchsensor
*Light Sensor
*Display
*Rotation Sensor
*Playing Sounds

Template for Robot programs

Here is a link to a template you can use when writing your own robot programs. Just click on the link below to open the template in Jcreator.
You have to save the file in **c:\UIO\web\programs**
Remember to change the on top of the file so that it matches the name of your file. Remember that is has to start with a capital letter

Template. Choose open and it will open in Jcreator

A complete robot program

Here is a link to a complete and working program.
Click on the link below to open the file in Jcreator
Be sure to save it in the folder **c:\UIO\web\programs\**

Complete Robot. **Choose open, and it will be opened with Jcreator**

Piece of code :: Using random number

- This is a piece of code to show you how to use a random number
- To use it just just copy it from here and paste it in your own program

```
        Assistant harry; // We want to have an assistant
        harry = new Assistant(); // Here we make the new assistant
        int waitseconds; // We make a variable called waitseconds. It can contain a
number
        // Putting a value in the waitseconds variable
        waitseconds = harry.randomNumber(5000);

        // Now waitseconds contains a value between 0 og 5000
```

Piece of code :: Wait and Legomotor

- Here is a piece of code to explain the use of Legomotor and
- how to set the time between each command
- Just cut the code out and paste it in your own program

```
     // The motor has to be made in the Robot class since it is a part of the Robot
    Legomotor rightMotor; // we want a rightMotor
    Legomotor leftMotor; // we want a leftMotor

    rightMotor = new Legomotor(A); // We connect this motor to port A on the RCX
    leftMotor = new Legomotor(C); // We connect this motor to port C on the RCX

     // This has to be pasted into the Control class since this is where we control
the robot

    Assistant harry; // We want an assistant called harry
    Robot volvo;

    harry = new Assistant(); // The new assistant is created
    volvo = new Robot(); // The new robot is created

    volvo.rightMotor.forward(7);  // The motor runs forward with power 7
    volvo.leftMotor.forward(7);
    harry.wait(2500); // harry asks the program to wait 2.5 seconds and then continue

     // The following methods explain themself
    volvo.rightMotor.stop();
    volvo.leftMotor.stop();
```

Piece of Code :: Using the touchsensor

- This is a piece of code on how to use the touch sensor
- Just cut it out and paste it into your code

```
  // This is part of the Robot
 Touchsensor touch1; // we want to use a touchsensor
 touch1 = new Touchsensor(1); // We connect this touchsensor to port 1


  // And this belongs in the Control class
  // you have to have already made a robot called volvo

 volvo.rightMotor.forward(7);
 volvo.leftMotor.forward(7);

 // as long as the touchsensor isn't pushed, just continue
 // when it is finally pushed, then get out of the while loop and continue program

 while(volvo.touch1.isPressed() == false){
 continue; // jumps to the top of the while loop
 }
 // here will the program go when the touchsensor is pushed
 volvo.rightMotor.stop();
 volvo.leftMotor.stop();
```

Piece of Code :: Advanced touch sensors

- This is an advanced piece of code for using a touch sensor
- It won't work if you haven't already made 2 touch sensors on the robot

```
while(true){ //this is a while loop that will run forever until we tell it to break;
//if the first touchsensor register a push then do what's inside the if sentence
        if(touch1.isPressed() == true){
                leftMotor.backward(7);
                rightMotor.backward(7);
                harry.wait(3000);
                break;// here we will jump out of the while loop.
        }
// if the second touchsensor registers a push, the result will be slightly different.
        if(touch2.isPressed() == true){
                leftMotor.backward(7);
                rightMotor.backward(7);

                harry.wait(500);
                break;

        }
//this is the end of the while loop. If neighter of the sensors are pushed, we will
check them again
//as the programs starts from the top of the while loop again.
//Forever until one of the sensors are pushed

}
```

Piece of Code :: Using a light sensor

- This is a piece of code on how to use the light sensor

```
     // This is part of the robot
    Lightsensor light1; // we want to use a lightsensor
    light1 = new Lightsensor(2); // we connect the new lightsensor to port 2

     // This belongs in the Control class since it controls the robot
     // You need a robot called volvo

    volvo.rightMotor.forward(7);
    volvo.leftMotor.forward(7);

    // this is a while loop that will run as long as the light sensor says that
the
    // luminosity is greater than 33
    while(volvo.light1.luminosity() > 33){
        continue;
    }

    volvo.rightMotor.stop();
    volvo.leftMotor.stop();
```

Main Pages:
*Lego and Java
*Resturant
Building Hints:
*Simple Vehichle
*Vehichle w/ belts
*Touch Sensors
Assignments:
*Touchsensor Task
*Black Line 1
*Black Line 2
*Sound Task
*Rotation Task
Teaching Material:
*Transfer Tool
*Using JCreator
*Template for Robots
*Complete Robot
Pieces of Code:
*Random Number.
*Motor and Wait.
*Touch Sensor
*Adv. Touchsensor
*Light Sensor
*Display
*Rotation Sensor
*Playing Sounds

Piece of Code :: Using the Display

- This is a piece of code to explain how to use the Display on the RCX

```
     // This is part of the robot
    Display lcd;  // We want a display called lcd on the robot
    lcd = new Display();  We make our new Display

     // This belongs in the Control class since it controls the robot
     // You need an Assistant named harry and a robot named volvo

    String textString; // We want a textString
    textString = "Math"; // I want the textstring to contain the word "Math"
    volvo.lcd.print(textString); // This prints the text to the Display
    harry.wait(3000); // This will be shown for 3 seconds before the program ends
```

Piece of Code :: How to use a rotation sensor

- This is a piece of code that explains the use of the rotation sensor

```
 // This is part of the robot
Rotationsensor rotation1; // we want a rotation sensor on our robot
rotation1 = new Rotationsensor(1); // we connect the rotation sensor to port 1
Display lcd;  // we want a display as well
lcd = new Display();  so we create a new display


 // This belongs in the Control class since it controls the robot
 // You need a robot named volvo and an assistant named harry

volvo.rotation1.reset(); // we want the counter of the rotation sensor to be
0
volvo.rightMotor.forward(7);
volvo.leftMotor.forward(7);
harry.wait(2500);

int rotationValue; // we make a variable called rotationValue
// Then we put the number of rotations into the rotationValue variable
rotationValue = volvo.rotation1.rotation();

volvo.rightMotor.stop();
volvo.leftMotor.stop();

volvo.lcd.print(rotationValue); // we print out the rotationvalue to the display
harry.wait(3000); // we show the value for 3 seconds before the program ends
```

Piece of Code :: How to make the RCX play music

- This is a piece of code to make the RCX play tones given a frequency and a lenght

```
     // This is part of the robot
    Speaker soundplayer;  // We want the robot to have a Speaker
    soundplayer = new Speaker();  We make a new Speaker

     // This belongs in the Control class since it controls the robot
     // it needs a robot called volvo to work

     // this soundplayer plays a tone with a frequency of 500Hz
     // and a length of 200/100 seconds = 2 seconds
    volvo.soundplayer.playTone(500,200);
```

```
import josx.platform.rcx.*;
import java.util.*;

class Pressuresensor {

    private Sensor sen;

    public Pressuresensor(int number) {
        if (number == 1)
            sen = Sensor.S1;
        if (number == 2)
            sen = Sensor.S2;
        if (number == 3)
            sen = Sensor.S3;

        sen.setTypeAndMode(1, 0x20);
    }

    public boolean isPressed() {
        return sen.readBooleanValue();
    }
}


class Lightsensor {

    private Sensor sen;

    public Lightsensor(int number) {
        if (number == 1)
            sen = Sensor.S1;
        if (number == 2)
            sen = Sensor.S2;
        if (number == 3)
            sen = Sensor.S3;

        sen.setTypeAndMode(3, 0x80);
        sen.activate();
    }

    public int luminosity() {
        return Math.abs(sen.readValue());
    }
}

class Rotationsensor {

        private Sensor sen;
        private int n;

    public Rotationsensor(int number) {
        n = number;
        if (number == 1)
            sen = Sensor.S1;
        if (number == 2)
            sen = Sensor.S2;
        if (number == 3)
            sen = Sensor.S3;

        sen.setTypeAndMode(4, 0xE0);
        sen.activate();
    }
```

```
    public void reset() {
        sen.setPreviousValue(0);
    }

    public int rotation() {
        return Math.abs(sen.readValue());
    }


}

class Temperaturesensor {

    private Sensor sen;

    public Temperaturesensor(int number) {
        if (number == 1)
            sen = Sensor.S1;
        if (number == 2)
            sen = Sensor.S2;
        if (number == 3)
            sen = Sensor.S3;

        sen.setTypeAndMode(2, 0xA0);
        sen.activate();
    }

    public void reset() {
        sen.setPreviousValue(0);
    }

    public int temperature() {
        //return sen.readSensorValue(n-1, 0);
        return Math.abs(sen.readValue());
    }


}

class Assistant{

    public Assistant(){}

    void wait(int msek) {
        try {
            Thread.sleep(msek);
        } catch(InterruptedException ie) {}
    }

    int randomNumber(int maxNum){
        double rnd = Math.random();
        double random = (double)maxNum * rnd;
        return (int) random;
    }
}

class Speaker {
    public Speaker() {}
    Assistant ast = new Assistant();

    void playNote(int freq, int length){
        Sound.playTone(freq, length);
        ast.wait(10*length);
    }
}

class Display {
    public Display() {}

    void write(String s){
        TextLCD.print(s);
    }

    void write(int t){
        LCD.showNumber(t);
    }
}

class Legomotor{

    Motor mot = null;

    public Legomotor(int number) {
        if (number == 1)
                mot = Motor.A;
        if (number == 2)
                mot = Motor.B;
        if (number == 3)
            mot = Motor.C;
    }

    void forward(int pow){
        mot.setPower(pow);
        mot.forward();
    }

    void backward(int pow){
        mot.setPower(pow);
        mot.backward();
    }

    void stop(){
        mot.stop();
    }
    void softStop(){
        mot.flt();
    }
}

class Light {

    Motor mot = null;

    public Light(int number) {
        if (number == 1)
            mot = Motor.A;
        if (number == 2)
            mot = Motor.B;
        if (number == 3)
            mot = Motor.C;
    }

    void turnOn(int pow){
```

```
        mot.setPower(pow);
        mot.forward();
    }

    void turnOff(){
        mot.stop();
    }
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Skybært2 {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.
Display ut;

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre
        ut = new Display();

        volvo.kneipen.reset();


        while(true) {

                volvo.venstreMotor.kjørFremover(5);
                volvo.høyreMotor.kjørFremover(5);

                ut.skriv(volvo.kneipen.rotasjon());

                if(volvo.my.lysstyrke()<43) {
                        volvo.venstreMotor.kjørBakover(6);

                        per.vent(100);
                        }
                if(volvo.mummi.lysstyrke()<43) {

                        volvo.høyreMotor.kjørBakover(6);

                }       per.vent(100);



                }




        }

    // idet trykksensor1 blir trykket inn, så hopper programmet hit

        }//her er programmet ferdig


//nå gjenstår det å lage selveste bilen
```

```

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Trykksensor ivar;
Trykksensor iver;
Lyssensor mummi;
Lyssensor my;
Speaker trond;
Rotasjonssensor kneipen;

//osv.....

        Robot(){
        //*******************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //*******************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCX
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        mummi = new Lyssensor(1);
        my = new Lyssensor(3);
        trond= new Speaker();
        kneipen= new Rotasjonssensor (2);

        }
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Gunnar {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.
        Kontroll(){
    Speaker lars;
    Display beate;
        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre
        lars = new Speaker();
        beate = new Display();

        /*
    volvo.venstreMotor.kjørFremover(7);
    volvo.høyreMotor.kjørFremover(7);
    per.vent(4000);
    lars.spillTone(560, 50);
    volvo.venstreMotor.kjørFremover(5);
    volvo.høyreMotor.kjørFremover(5);
    per.vent(3500);

    while(volvo.kurtkåre.lysstyrke() > 40) {
        volvo.høyreMotor.kjørFremover(6);
        volvo.venstreMotor.kjørFremover(6);
    }

    per.vent(4000);
    lars.spillTone(738, 50);
    volvo.venstreMotor.kjørBakover(2);
    volvo.høyreMotor.kjørFremover(3);
    per.vent(2000);
    lars.spillTone(440, 70);
    lars.spillTone(536, 30);
    lars.spillTone(694, 48);
    volvo.venstreMotor.kjørFremover(4);
    volvo.høyreMotor.kjørFremover(4);
    volvo.snu();
    per.vent(3000);
    beate.skriv("hallo");
    lars.spillTone(460, 73);
    beate.skriv("marit");




        volvo.høyreMotor.kjørFremover(4);
        volvo.venstreMotor.kjørFremover(4);

   if   (volvo.kurtkåre.lysstyrke() < 40) {
```

```
        volvo.venstreMotor.kjørBakover(3);
        volvo.høyreMotor.kjørBakover(3);
        per.vent(1000);
        volvo.snu();
           }

    */
        while (true) {
    if (volvo.kårekurt.lysstyrke() < 40) {
        volvo.venstreMotor.kjørFremover(4);
        volvo.høyreMotor.kjørFremover(4);
        }
    else {
        volvo.høyreMotor.kjørBakover(3);
        volvo.venstreMotor.kjørBakover(3);



    if (volvo.kurtkåre.lysstyrke() < 40) {
        volvo.høyreMotor.kjørFremover(3);
        volvo.venstreMotor.kjørBakover(3);
        }

   if  (volvo.kårekurt.lysstyrke() < 40) {
        volvo.venstreMotor.kjørFremover(3);
        volvo.høyreMotor.kjørBakover(3);
    }

    }



    }
    }

        //her er programmet ferdig
}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Lyssensor kurtkåre;
Assistent nils;
Lyssensor kårekurt;
//osv.....

        Robot(){
        //*******************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //*******************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCX
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        kurtkåre = new Lyssensor (1);
        nils = new Assistent();
        kårekurt = new Lyssensor (3);
        }

        public void snu() {
                høyreMotor.kjørFremover(3);
                venstreMotor.kjørBakover(3);
                nils.vent(1500);
        }


}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt. Ved siden av "class" skriver du hva du vil robot
en skal hete.
class sasha {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot sasha ; //Her sier vi hva den nye roboten skal hete.
Assistent merethe; //Her sier vi hva ssistenten skal hete.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        sasha = new Robot();
        //og så lager vi en ny assistent:
        merethe = new Assistent();
        //under programmerer man roboten.

        sasha.høyreMotor.kjørFremover(6);
        sasha.venstreMotor.kjørFremover(6);
        sasha.venstreMotor.stopp();
        sasha.høyreMotor.stopp() ;
        sasha.venstreMotor.kjørFremover (4);
        sasha.høyreMotor.kjørFremover (4);
        //her sier du noe roboten skal gjøre hele tiden
        while(sasha.lys.lysstyrke() > 40) {
                merethe.vent(3000);
                sasha.venstreMotor.kjørFremover(5);
                sasha.høyreMotor.kjørFremover(5);

        }

        }//her er programmet ferdig

}
class Robot{
        //først må vi si hva roboten består av

        Legomotor venstreMotor;
        Legomotor høyreMotor;
        Lyssensor lys;
        Lyssensor lysrosa;

        // osv...

        Robot(){
                // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1; int B = 2; int C = 3;

                //her lager vi delene på bilen
                //en motor festet til port A på RCXen
                venstreMotor = new Legomotor(A);
                //en motor festet til port C på RCXen
                høyreMotor = new Legomotor(C);
                // osv...
                lys = new Lyssensor (1);
```

```
                lysrosa = new Lyssensor (3);





        }
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Arsenal4 {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot arsenal; //her sier vi at vi skal ha en ny robot som vi kaller arsenal.
Assistent bob; //vi vil ha med en assistent som heter bob.
Display ut;
        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        arsenal = new Robot();
        //og så lager vi en ny assistent:
        bob = new Assistent();
        //her kan dere skrive hva roboten skal gjøre

        ut = new Display();


        int gå = 0;

        while(gå == 0){

        arsenal.venstremotor.kjørFremover(6);
    arsenal.høyremotor.kjørFremover(6);

        if (arsenal.vensen.trykkesInn()==true){
                        arsenal.venstremotor.kjørBakover(5);
                        arsenal.høyremotor.kjørBakover(5);
                        bob.vent(2000);
                        arsenal.venstremotor.stopp();
                        bob.vent(1500);
                        }

        if (arsenal.høysen.trykkesInn()==true) {
                        arsenal.venstremotor.kjørBakover(5);
                        arsenal.høyremotor.kjørBakover(5);
                    bob.vent(2000);
                        arsenal.venstremotor.stopp();
                        bob.vent(1500);

                }


        if (arsenal.light.lysstyrke() < 41){
                        arsenal.venstremotor.stopp();
                        arsenal.høyremotor.stopp();
                        bob.vent(1000);
                    arsenal.venstremotor.kjørBakover(5);
                        arsenal.høyremotor.kjørBakover(5);
                    bob.vent(2000);
                        arsenal.venstremotor.stopp();
                        bob.vent(1500);

        }
```

```


        }//slutt while






        }//her er programmet ferdig

}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstremotor;
Legomotor høyremotor;
Trykksensor vensen, høysen;
Lyssensor light;

//osv.....

        Robot(){
        //*****************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //*****************************************************************


        //her lager vi alt som roboten skal bestå av
        venstremotor = new Legomotor(A);//en motor festet til port A på RCX
        høyremotor = new Legomotor(C);//en motor festet til port C på RCXen
        vensen = new Trykksensor(1);
        høysen = new Trykksensor(2);
        light = new Lyssensor(3);
        //osv.....

        }
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Donnie {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre

        /*per.vent(1500);
        volvo.stopp();
        volvo.snu();
        volvo.venstreMotor.kjørBakover(5);
        volvo.høyreMotor.kjørBakover(5);
        per.vent(1000);
        */
          //hehehe :D ,,,, dette er jo litt kult da =););)!!!!!! my kommentarer da
 liksom.. dette blir som en videodagbok hehe

        int tall;

        tall = 45;

        while (tall == 45){

                volvo.venstreMotor.kjørFremover(1);
            volvo.høyreMotor.kjørFremover(1);


                volvo.molle.skriv(volvo.kaare.lysstyrke());
                if((volvo.kaare.lysstyrke()<34) ){
                        volvo.høyreMotor.kjørFremover(1);
                        volvo.venstreMotor.kjørBakover(2);
                        per.vent(200);
                }

                if ((volvo.kjell.lysstyrke()<34) ) {
                        volvo.venstreMotor.kjørFremover(1);
                        volvo.høyreMotor.kjørBakover(2);
                        per.vent(200);
                }




        else {volvo.høyreMotor.kjørFremover(3);
        volvo.venstreMotor.kjørFremover(3);
```

```
        }


        }

        }//her er programmet ferdig

}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Assistent Alex;
Lyssensor kaare;
Lyssensor kjell;
Display molle;
//osv.....

        Robot(){
        //*******************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //*******************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCX
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        Alex = new Assistent();//osv.....
        kaare = new Lyssensor (1);
        kjell = new Lyssensor (3);
        molle = new Display ();
        }

        public void snu() {
                høyreMotor.kjørFremover(7);
                Alex.vent(900);

        }
    public void stopp() {
        høyreMotor.stopp();
        venstreMotor.stopp();
    }
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Arnold {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre

        volvo.venstreMotor.kjørFremover(7);
        volvo.høyreMotor.kjørFremover(7);
        per.vent(4000);

}
}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Lyssensor lyssensor1;
Lyssensor lyssensor2;
Display display1;
//osv.....

        Robot(){
        //**********************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //**********************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCXen
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
    lyssensor1 = new Lyssensor(1);
        lyssensor2 = new Lyssensor(3);
}
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Streken {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre


        volvo.høyreMotor.kjørFremover(4);
        volvo.venstreMotor.kjørFremover(4);

        boolean temp = true;

        while (temp){
                if(volvo.lys1.lysstyrke() < 37)
                        volvo.svingH();
                if(volvo.lys3.lysstyrke() < 37)
                        volvo.svingV();
        }

        }//her er programmet ferdig

}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Lyssensor lys1;
Lyssensor lys3;
Assistent ola;

        Robot(){
        //**********************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //**********************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCXen
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        lys1 = new Lyssensor(1);
```

```
        lys3 = new Lyssensor(3);
        ola = new Assistent();

        }

        void svingV(){
//              venstreMotor.kjørBakover(5);
                høyreMotor.kjørBakover(4);
//              ola.vent(1000);
//              venstreMotor.kjørFremover(5);
                ola.vent(100);
                høyreMotor.kjørFremover(4);

        }

        void svingH(){
                venstreMotor.kjørBakover(4);
//              høyreMotor.kjørBakover(5);
//              ola.vent(1000);
//              høyreMotor.kjørFremover(5);
                ola.vent(100);
                venstreMotor.kjørFremover(4);

        }


}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Snurre2 {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot snurre; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        snurre = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre


        while(true){

        snurre.venstreMotor.kjørFremover(7);
        snurre.høyreMotor.kjørFremover(7);

        if (snurre.trykksensor1.trykkesInn()==true){
        snurre.venstreMotor.kjørBakover(7);
        snurre.høyreMotor.kjørBakover(7);
        per.vent(1000);
        snurre.venstreMotor.kjørFremover(7);
        per.vent(500);
        }


        if (snurre.trykksensor3.trykkesInn()==true){
        snurre.venstreMotor.kjørBakover(7);
        snurre.høyreMotor.kjørBakover(7);
        per.vent(1000);
        snurre.høyreMotor.kjørFremover(7);
        per.vent(500);

        }

        if (snurre.trykksensor2.trykkesInn()==true){
        snurre.midtMotor.kjørBakover(7);
        per.vent(1200);
        snurre.midtMotor.stopp();


        }

    }

    // idet trykksensor1 blir trykket inn, så hopper programmet hit


        }//her er programmet ferdig
}
```

```

//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Trykksensor trykksensor1;
Trykksensor trykksensor3; // vi skal nå ha med en trykksensor
Trykksensor trykksensor2;
Speaker lyd;
Legomotor midtMotor;
//osv.....

        Robot(){
                //***********************************************************
*******
                // Helpevariabler for motorer. Ikke bry dere om disse
                int A = 1;
                int B = 2;
                int C = 3;
                //***********************************************************
*******

                //her lager vi alt som roboten skal bestå av
                venstreMotor = new Legomotor(A);//en motor festet til port 
RCXen
                høyreMotor = new Legomotor(C);//en motor festet til port C 
Xen
                trykksensor1 = new Trykksensor(1); // denne nye trykksensor
al vi ha på port 1
                trykksensor3 = new Trykksensor(3);
                trykksensor2 = new Trykksensor(2);
                lyd = new Speaker();
                midtMotor = new Legomotor(B);


        }

        public void snu() {
                høyreMotor.kjørFremover(5);
                venstreMotor.kjørBakover(5);

        }


}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Bae {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot jack; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        jack= new Robot();
        //og så lager vi en ny assistent:
        per= new Assistent();
        //her kan dere skrive hva roboten skal gjøre

        /*
        jack.venstreMotor.kjørFremover(7);
        jack.høyreMotor.kjørFremover(7);
        per.vent(1000);
        jack.høyreMotor.kjørBakover(5);
        jack.venstreMotor.kjørFremover(5);
        per.vent(1000);
        jack.høyreMotor.stopp();
        jack.venstreMotor.stopp();
        per.vent(1000);
        jack.venstreMotor.kjørBakover(4);
        jack.høyreMotor.kjørBakover(5);
        per.vent(1500);
        jack.høyreMotor.stopp();
        jack.venstreMotor.stopp();
        per.vent(500);
        jack.høyreMotor.kjørFremover(6);
        jack.venstreMotor.kjørBakover(3);
        jack.lars.spillTone(500, 125);
        per.vent(3000);
        jack.lars.spillTone(2000, 240);
        jack.høyreMotor.kjørFremover(7);
        jack.venstreMotor.stopp();
        per.vent(6000);
        jack.lars.spillTone(1200, 240);
        jack.lars.spillTone(21, 100);
        jack.høyreMotor.kjørFremover(7);
        jack.venstreMotor.kjørBakover(7);
        per.vent(3000);
        jack.høyreMotor.kjørFremover(7);
        jack.venstreMotor.kjørFremover(6);
        jack.lars.spillTone(43, 300);
        per.vent(3000);
        jack.høyreMotor.kjørBakover(7)
        jack.venstreMotor.kjørBakover(2)
        jack.lars.spillTone(2000, 200)
        per.vent(2000)
        jack.høyremotor.kjørFremover(4)
        jack.venstremotor.kjørFremover(4)
```

```
        per.vent(500)



        int tall;
        tall = 1;

        while(tall == 1){
          jack.høyreMotor.kjørFremover(5);
          jack.venstreMotor.kjørFremover(5);
          if (jack.eye.lysstyrke() < 33){
            jack.venstreMotor.kjørFremover(5);
            jack.høyreMotor.kjørBakover(5);
            per.vent(500);
            if (jack.kjel.lysstyrke() < 33){
              jack.høyreMotor.kjørFremover(5);
              jack.venstreMotor.kjørBakover(5);
              per.vent(500);
              if (jack.eye.lysstyrke() < 33){
                jack.høyreMotor.kjørFremover(5);
                jack.venstreMotor.kjørFremover(5);
              }

            }
          }

        jack.kran.kjørFremover(7);
        per.vent(4000);
        jack.kran.kjørFremover(7);
        jack.høyreMotor.kjørFremover(6);
        jack.venstreMotor.kjørFremover(5);
        per.vent(4000);
        jack.høyreMotor.kjørFremover(5);
        jack.venstreMotor.kjørBakover(4);
        per.vent(3000);
        jack.høyreMotor.kjørBakover(6);
        jack.venstreMotor.kjørBakover(6);
        per.vent(1000);
        jack.høyreMotor.kjørFremover(7);
        per.vent(2000);
        jack.høyreMotor.kjørFremover(7);
        jack.venstreMotor.kjørFremover(7);
        per.vent(4000);
        jack.kran.kjørFremover(7);
        per.vent(2400);
        jack.venstreMotor.kjørFremover(5);
        jack.høyreMotor.kjørFremover(3);
        jack.venstreMotor.kjørFremover(2);
        jack.høyreMotor.kjørFremover(6);
        jack.lars.spillTone(21,2000);
        jack.lars.spillTone(21,1500);
        jack.lars.spillTone(21,1000);
        jack.lars.spillTone(21,500);
        per.vent(3000);
        jack.venstreMotor.kjørFremover(4);
        per.vent(2000);

        jack.høyreMotor.kjørBakover(4);
        jack.kranåpning.kjørFremover(6);
        jack.kranåpning.kjørBakover(3);
        if(jack.eye.lysstyrke() < 40){
        jack.venstreMotor.kjørFremover(7);
        jack.
        */
        while (true) {
                jack.venstreMotor.kjørFremover(7);
                jack.høyreMotor.kjørFremover(7);

                if (jack.kjel.lysstyrke() < 37){
                        jack.kran.kjørFremover(7);
                        per.vent(2000);
                        jack.venstreMotor.kjørFremover(7);
                        jack.høyreMotor.kjørFremover(1);
                        per.vent(1000);
                        jack.høyreMotor.kjørFremover(5);
                        jack.venstreMotor.kjørBakover(2);
                        per.vent(4000);
                        }

        }

        }


}



        //her er programmet ferdig


//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Speaker lars;
Lyssensor eye;
Lyssensor kjel;
Legomotor kran;
Legomotor kranåpning;
//osv.....

        Robot(){
        //*********************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //*********************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCXen
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        lars = new Speaker();
        eye = new Lyssensor(1);
        kjel = new Lyssensor(3);
        kran = new Legomotor(B);
        kranåpning = new Legomotor(2);
        //osv.....

        }
```

```
}
```

```
// Disse to linjene trengs for å fortelle programmet hvilke verktøy vi trenger
import josx.platform.rcx.*;
import java.util.*;

// Her starter programmet vårt.
class Kandoo {
    public static void main(String [] args) {
        new Kontroll();
    }
}

class Kontroll {
Robot volvo; //her sier vi at vi skal ha en ny robot som vi kaller volvo.
Assistent per; //vi vil ha med en assistent som heter per.

        Kontroll(){

        //her lager vi en ny bil som vi skal programmere:
        volvo = new Robot();
        //og så lager vi en ny assistent:
        per = new Assistent();
        //her kan dere skrive hva roboten skal gjøre

        volvo.kjør(2);
        per.vent(2000);

        volvo.bakover(2);
        per.vent(555);

        volvo.kjør(2);
        per.vent(2000);

        volvo.svinghøyre(3,2100);
        volvo.stans();

        volvo.svingvenstre(2,2000);
        volvo.stans();
        //volvo.Louie.spillTone(1100, 250);


     //while (volvo.Belly.lysstyrke()>35) {
        //      volvo.kjør(3);
        //}

        //volvo.Anna.skriv("strek");
        //per.vent(1000);

        //volvo.kjør(1)
        //per.vent(1500)





        //volvo.Anna.skriv("Marit");
        //per.vent(2600);



        //volvo.Anna.skriv(volvo.Olly.lysstyrke());
        //per.vent(1000);
```

```


        //volvo.Louie.spillTone(800, 180);
        //volvo.Louie.spillTone(1000, 200);

        }//her er programmet ferdig

}
//nå gjenstår det å lage selveste bilen

class Robot{
//først må vi si hva roboten består av
Legomotor venstreMotor;
Legomotor høyreMotor;
Speaker Louie;
Display Anna;
Lyssensor Belly;
Lyssensor Olly;
Assistent petter;
Robot(){
        //******************************************************************
        // Helpevariabler for motorer. Ikke bry dere om disse
        int A = 1;
        int B = 2;
        int C = 3;
        //******************************************************************


        //her lager vi alt som roboten skal bestå av
        venstreMotor = new Legomotor(A);//en motor festet til port A på RCX
        høyreMotor = new Legomotor(C);//en motor festet til port C på RCXen
        Louie = new Speaker();
        Anna = new Display(); //osv.....
        Belly = new Lyssensor(1);
        Olly = new Lyssensor(3);
        petter = new Assistent();

        }

        public void kjør(int styrke) {
                høyreMotor.kjørFremover(styrke);
                venstreMotor.kjørFremover(styrke);
        }


        public void stans() {
                høyreMotor.stopp();
                venstreMotor.stopp();
        }

        public void bakover(int styrke){
                høyreMotor.kjørBakover(styrke);
                venstreMotor.kjørBakover(styrke);
        }

        public void svinghøyre(int styrke, int tid) {
                venstreMotor.kjørFremover(styrke);
                høyreMotor.stopp();
                petter.vent(tid);
        }
        public void svingvenstre(int styrke, int tid) {
                høyreMotor.kjørFremover(styrke);
                venstreMotor.stopp();
                petter.vent(tid);
        }

}
```

```


class Kontroll {

    Resturant vertshuset;
    Person per;
    Person lise; // osv
        Person kjell;
        Person hanne;

    Kontroll() {
        vertshuset = new Resturant();

        // Under her kan dere skrive kommandoene deres.
                per = new Gjest("Per", "mann");
                lise = new Gjest ("lise","kvinne");
                kjell = new Gjest("kjell","mann");
                hanne =new Gjest("hanne","kvinne");

                vertshuset.hovmester.endreAntallPlasser(vertshuset.bord4,4);

                vertshuset.hovmester.plasserVedBord(per, vertshuset.bord4);
                vertshuset.hovmester.plasserVedBord(lise, vertshuset.bord4);
                vertshuset.hovmester.plasserVedBord(kjell, vertshuset.bord4);
                vertshuset.hovmester.plasserVedBord(hanne, vertshuset.bord4);

                per.velgFraMeny(vertshuset.meny.rettNummer("Biff"),vertshuset.men
y.rettNummer("Kake"));
                lise.velgFraMeny(vertshuset.meny.rettNummer("Salat"),vertshuset.m
eny.rettNummer("Gele"));
                kjell.velgFraMeny(vertshuset.meny.rettNummer("Laks"),vertshuset.
meny.rettNummer("Sjokoladepudding"));
                hanne.velgFraMeny(vertshuset.meny.rettNummer("Salat"),vertshuset.
meny.rettNummer("Is"));


                vertshuset.bord4.servitør.taBestilling(per);
                vertshuset.bord4.servitør.taBestilling(lise);
                vertshuset.bord4.servitør.taBestilling(kjell);
                vertshuset.bord4.servitør.taBestilling(hanne);

                vertshuset.kokk.lageMat();
                vertshuset.kokk.lageMat();
                vertshuset.kokk.lageMat();
                vertshuset.kokk.lageMat();

                vertshuset.bord4.servitør.hentHovedrett();

                vertshuset.bord4.servitør.leverMatrett(vertshuset.bord4.servitør
.finnRiktigGjest(vertshuset.bord4));
                vertshuset.bord4.servitør.hentHovedrett();
                vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
                vertshuset.bord4.servitør.leverMatrett(vertshuset.bord4.servitør
.finnRiktigGjest(vertshuset.bord4));
                vertshuset.bord4.servitør.hentHovedrett();
                vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
                vertshuset.bord4.servitør.leverMatrett(vertshuset.bord4.servitør
.finnRiktigGjest(vertshuset.bord4));
                vertshuset.bord4.servitør.hentHovedrett();
                vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
                vertshuset.bord4.servitør.leverMatrett(vertshuset.bord4.servitør
```

```
.finnRiktigGjest(vertshuset.bord4));







    }
}


class Resturant {

    Kelner hovmester, bob;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bob = new Kelner("bob", "mann");
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person ole;
    Person kari;
    Person siri; //osv
    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("per", "mann");
        ole = new Gjest("ole", "mann");
        kari = new Gjest("kari", "dame");
        siri = new Gjest("siri", "dame");
        // osv



        // Under her kan dere skrive kommandoene deres.
        vertshuset.hovmester.plasserVedBord(per, vertshuset.bord1);
        vertshuset.hovmester.plasserVedBord(siri, vertshuset.bord1);
        vertshuset.hovmester.plasserVedBord(kari, vertshuset.bord1);
        vertshuset.hovmester.plasserVedBord(ole, vertshuset.bord1);
        per.velgFraMeny(0, 10);
    ole.velgFraMeny(4, 11);
    kari.velgFraMeny(5, 6);
    siri.velgFraMeny(3, 6);
        vertshuset.bord1.servitør.taBestilling(per);
    vertshuset.bord1.servitør.taBestilling(ole);
    vertshuset.bord1.servitør.taBestilling(kari);
    vertshuset.bord1.servitør.taBestilling(siri);
    vertshuset.tor.lageMat();
    vertshuset.bord1.servitør.hentHovedrett();
    vertshuset.bord1.servitør.hentHovedrett();
    vertshuset.bord1.servitør.hentHovedrett();
    vertshuset.bord1.servitør.hentHovedrett();
    vertshuset.bord1.servitør.hentDessert();
    vertshuset.bord1.servitør.hentDessert();
    vertshuset.bord1.servitør.hentDessert();
    vertshuset.bord1.servitør.hentDessert();

    }
}


class Resturant {

    Kelner hovmester;
        Kokk tor;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        tor = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
```

```
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(4);
    }
}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person lars;
    Person nina;
    Person hanne; // osv
    Person gjest;

    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("Per", "mann");
        lars = new Gjest("Lars", "mann");
        nina = new Gjest("Nina", "kvinne");
        hanne = new Gjest("Hanne", "kvinne");
        gjest = new Gjest ("Kalle", "mann");
        // osv


        // Under her kan dere skrive kommandoene deres.

    vertshuset.hovmester.endreAntallPlasser(vertshuset.bord4,4);
    vertshuset.hovmester.plasserVedBord(per, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(lars,vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(nina,vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(hanne,vertshuset.bord4);
    per.velgFraMeny(vertshuset.meny.rettNummer("Biff"),vertshuset.meny.rettNummer
("Gele"));
    lars.velgFraMeny(vertshuset.meny.rettNummer("Laks"),vertshuset.meny.rettNumm
er("Is"));
    nina.velgFraMeny(vertshuset.meny.rettNummer("Salat"),vertshuset.meny.rettNumm
er("Is"));
    hanne.velgFraMeny(vertshuset.meny.rettNummer("Biff"),vertshuset.meny.rettNumm
er("Sjokoladepudding"));
    vertshuset.bord4.servitør.taBestilling(per);
    vertshuset.bord4.servitør.taBestilling(lars);
    vertshuset.bord4.servitør.taBestilling(nina);
    vertshuset.bord4.servitør.taBestilling(hanne);
    vertshuset.kokk.lageMat();
    vertshuset.bord4.servitør.hentHovedrett();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentHovedrett();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentHovedrett();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentHovedrett();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentDessert();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentDessert();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentDessert();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
```

```
    vertshuset.bord4.servitør.leverMatrett(gjest);
    vertshuset.bord4.servitør.hentDessert();
    gjest = vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(gjest);




    }
}


class Resturant {

    Kelner hovmester;
    Kelner kelner;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kelner = new Kelner("Hans", "mann");
        kelner = new Kelner("Grete", "kvinne");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant mia;
    Person arne;
    Person bjarne; // osv
    Person chris;
    Person daniel;

    Kontroll() {
        mia = new Resturant();
        arne = new Gjest("1", "mann");
        bjarne = new Gjest("2", "mann");
        chris = new Gjest("3", "mann");// osv
        daniel = new Gjest("4", "mann");


      mia.hovmester.endreAntallPlasser(mia.bord4, 4);


      mia.hovmester.plasserVedBord(arne, mia.bord4);
      mia.hovmester.plasserVedBord(bjarne, mia.bord4);
      mia.hovmester.plasserVedBord(chris, mia.bord4);
      mia.hovmester.plasserVedBord(daniel, mia.bord4);

       arne.velgFraMeny(1, 1);
    bjarne.velgFraMeny(2, 2);
        chris.velgFraMeny(3, 3);
      daniel.velgFraMeny(4, 5);


      mia.bord4.servitør.taBestilling(arne);
      mia.bord4.servitør.taBestilling(bjarne);
      mia.bord4.servitør.taBestilling(chris);
      mia.bord4.servitør.taBestilling(daniel);



      mia.kokk.lageMat();


      mia.bord4.servitør.hentHovedrett();
    mia.bord4.servitør.hentHovedrett();
   mia.bord4.servitør.hentHovedrett();
   mia.bord4.servitør.hentHovedrett();




       mia.kelner.finnRiktigGjest(4);



       mia.kelner.leverMatrett(arne);
       mia.kelner.leverMatrett(bjarne);
       mia.kelner.leverMatrett(chris);
       mia.kelner.leverMatrett(daniel);


       mia.bord4.servitør.hentDessert();
```

```
       mia.bord4.servitør.hentDessert();
       mia.bord4.servitør.hentDessert();
       mia.bord4.servitør.hentDessert();


      mia.kelner.finnRiktigGjest(4);


      mia.kelner.leverMatrett(arne);
       mia.kelner.leverMatrett(bjarne);
       mia.kelner.leverMatrett(chris);
       mia.kelner.leverMatrett(daniel);





        // Under her kan dere skrive kommandoene deres.



    }
}


class Resturant {

    Kelner hovmester;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person lise; // osv
    Person ole;
    Person karl;

    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("Per", "mann");
        lise = new Gjest("Lise", "dame");
        ole = new Gjest("Ole", "mann");
        karl = new Gjest("Karl", "mann");

        // osv

        // Under her kan dere skrive kommandoene deres.
    vertshuset.hovmester.endreAntallPlasser(vertshuset.bord4, 4);

    vertshuset.hovmester.plasserVedBord(per, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(lise, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(ole, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(karl, vertshuset.bord4);

    per.velgFraMeny(vertshuset.meny.rettNummer("Elgstek"),vertshuset.meny.rettNumm
er("Bær"));
    lise.velgFraMeny(vertshuset.meny.rettNummer("Kotelett"),vertshuset.meny.rettNu
mmer("Is"));
    ole.velgFraMeny(vertshuset.meny.rettNummer("Salat"),vertshuset.meny.rettNumme
r("Gele"));
    karl.velgFraMeny(vertshuset.meny.rettNummer("Oksestek"),vertshuset.meny.rettN
ummer("Kake"));


    vertshuset.bord4.servitør.taBestilling(per);
    vertshuset.bord4.servitør.taBestilling(lise);
    vertshuset.bord4.servitør.taBestilling(ole);
    vertshuset.bord4.servitør.taBestilling(karl);

    vertshuset.kokk.lageMat();

    vertshuset.bord4.servitør.hentHovedrett();

    vertshuset.bord4.servitør.hentDessert();

    vertshuset.bord4.servitør.finnRiktigGjest(vertshuset.bord4);
    vertshuset.bord4.servitør.leverMatrett(per);
    vertshuset.bord4.servitør.leverMatrett(lise);
    vertshuset.bord4.servitør.leverMatrett(ole);
    vertshuset.bord4.servitør.leverMatrett(karl);

    }
}


class Resturant {

    Kelner hovmester;
```

```
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person lise; // osv
        Person kjell;
        Person hanne;
    Display skjerm;
    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("Per", "mann");
        lise = new Gjest("lise", "dame");
        kjell = new Gjest("kjell", "mann");
        hanne = new Gjest("hanne", "dame");
        skjerm = new Display();
        // Under her kan dere skrive kommandoene deres.

        skjerm.skriv("velkommem");
        vertshuset.hovmester.endreAntallPlasser(vertshuset.bord4, 4);
        vertshuset.hovmester.plasserVedBord(lise, vertshuset.bord4);
        vertshuset.hovmester.plasserVedBord(kjell, vertshuset.bord4);
        vertshuset.hovmester.plasserVedBord(hanne, vertshuset.bord4);
        vertshuset.hovmester.plasserVedBord(per, vertshuset.bord4);

        lise.velgFraMeny(vertshuset.meny.rettNummer("Salat"), vertshuset.meny.ret
tNummer("Sjokoladepudding"));
        kjell.velgFraMeny(vertshuset.meny.rettNummer("Biff"), vertshuset.meny.ret
tNummer("Is"));
        hanne.velgFraMeny(vertshuset.meny.rettNummer("Elgstek"), vertshuset.meny.r
ettNummer("Tiramisu"));
        per.velgFraMeny(vertshuset.meny.rettNummer("Laks"), vertshuset.meny.rett
Nummer("Kake"));
        vertshuset.servitrise.taBestilling(per);
        vertshuset.servitrise.taBestilling(lise);
        vertshuset.servitrise.taBestilling(hanne);
        vertshuset.servitrise.taBestilling(kjell);
        vertshuset.kokk.lagMat();
        vertshuset.servitrise.hentHovedrett();
        skjerm.skriv("vaersaagod");
        skjerm.skriv("takk");
        vertshuset.servitrise.hentDessert();




    }
}


class Resturant {

    Kelner hovmester;
    Kelner servitrise;
    Kelner servitør;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
```

```
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        servitrise = new Kelner("Maren", "dame");
        servitør = new Kelner("Bjarne", "mann");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(3);
    }
    }
```

```
class Kontroll {

    Resturant oblecta;
    Person per;
    Person lise; // osv
        Person kjell;
        Person hanne;
    Person gjest;

    Kontroll() {
        oblecta = new Resturant();
        per = new Gjest("Per", "mann");
        lise = new Gjest("lise", "dame");
        kjell = new Gjest("kjell", "mann");
        hanne = new Gjest("hanne", "dame");
        gjest = new Gjest("gjest", "dame");

        // Under her kan dere skrive kommandoene deres.

        oblecta.hovmester.endreAntallPlasser(oblecta.bord4,4);
        oblecta.hovmester.plasserVedBord(per,oblecta.bord4);
        oblecta.hovmester.plasserVedBord(lise,oblecta.bord4);
        oblecta.hovmester.plasserVedBord(kjell,oblecta.bord4);
        oblecta.hovmester.plasserVedBord(hanne,oblecta.bord4);
        per.velgFraMeny(oblecta.meny.rettNummer("Elgstek"),oblecta.meny.rettNummer
("is"));
        lise.velgFraMeny(oblecta.meny.rettNummer("laks"),oblecta.meny.rettNummer(
"tiramisu"));
        kjell.velgFraMeny(oblecta.meny.rettNummer("kotelett"),oblecta.meny.rettNumm
er("sjokoladepudding"));
        hanne.velgFraMeny(oblecta.meny.rettNummer("salat"),oblecta.meny.rettNumme
r("gele"));
        oblecta.hovmester.taBestilling(per);
        oblecta.hovmester.taBestilling(lise);
        oblecta.hovmester.taBestilling(kjell);
        oblecta.hovmester.taBestilling(hanne);
        oblecta.kokk.lageMat();
        oblecta.hovmester.hentHovedrett();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
        oblecta.hovmester.hentHovedrett();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
        oblecta.hovmester.hentHovedrett();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
        oblecta.hovmester.hentHovedrett();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);

        oblecta.hovmester.hentDessert();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
        oblecta.hovmester.hentDessert();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
        oblecta.hovmester.hentDessert();
        gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
        oblecta.hovmester.leverMatrett(gjest);
```

```
        oblecta.hovmester.hentDessert();
            gjest = oblecta.hovmester.finnRiktigGjest(oblecta.bord4);
            oblecta.hovmester.leverMatrett(gjest);







    }
}


class Resturant {

    Kelner hovmester;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person lise; // osv
        Person kjell;
        Person hanne;

    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest ("Per", "mann");
        lise = new Gjest ("Lise", "dame");
        kjell = new Gjest ("Kjell", "mann");
        hanne = new Gjest ("Hanne", "dame");
                // Under her kan dere skrive kommandoene deres.
    vertshuset.hovmester.endreAntallPlasser(vertshuset.bord4,4);
    vertshuset.hovmester.plasserVedBord(per, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(lise, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(kjell, vertshuset.bord4);
    vertshuset.hovmester.plasserVedBord(hanne, vertshuset.bord4);
    vertshuset.meny.rettNummer("Biff");
    vertshuset.meny.rettNummer("Elgstek");
    vertshuset.meny.rettNummer("Salat");
    vertshuset.meny.rettNummer("Laks");
    vertshuset.meny.rettNummer("Kotelett");
    vertshuset.meny.rettNummer("Oksestek");
    vertshuset.meny.rettNummer("Is");
    vertshuset.meny.rettNummer("Kake");
    vertshuset.meny.rettNummer("Bær");
    vertshuset.meny.rettNummer("Sjokoladepudding");
    vertshuset.meny.rettNummer("Gele");
    vertshuset.meny.rettNummer("Tiramisu");
    per.velgFraMeny(0, 6);
    lise.velgFraMeny(1, 7);
    kjell.velgFraMeny(2, 8);
    hanne.velgFraMeny(3, 9);
    vertshuset.ida.taBestilling(per);
    vertshuset.ida.taBestilling(lise);
    vertshuset.ida.taBestilling(kjell);
    vertshuset.ida.taBestilling(hanne);
    vertshuset.kokk.lageMat();
    vertshuset.ida.hentHovedrett();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(per);
    vertshuset.ida.hentHovedrett();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(lise);
    vertshuset.ida.hentHovedrett();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(kjell);
    vertshuset.ida.hentHovedrett();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(hanne);
    vertshuset.ida.hentDessert();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(per);
    vertshuset.ida.hentDessert();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(lise);
```

```
    vertshuset.ida.hentDessert();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(kjell);
    vertshuset.ida.hentDessert();
    vertshuset.ida.finnRiktigGjest(vertshuset.bord4);
    vertshuset.ida.leverMatrett(hanne);
    }
}


class Resturant {

    Kelner hovmester;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;
    Kelner ida;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    ida = new Kelner("Ida","dame");
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person blabla; // osv

    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("Per", "mann");
        // osv



        // Under her kan dere skrive kommandoene deres.



    }
}


class Resturant {

    Kelner hovmester;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;

    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Tor", "mann");
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(2);
    }

}
```

```
class Kontroll {

    Resturant vertshuset;
    Person per;
    Person ole;
    Person pamula;
    Person erik;
    Person elise;

    Kontroll() {
        vertshuset = new Resturant();
        per = new Gjest("Per", "mann");
        ole = new Gjest("Ole", "mann");
         pamula = new Gjest("Pamula", "dame");
         erik = new Gjest("Erik", "mann");
          elise = new Gjest("Elise", "dame");
          // osv



        // Under her kan dere skrive kommandoene deres.

        vertshuset.hovmester.endreAntallPlasser(bord, 5);
        vertshuset.hovmester.plasserVedBord(per,bord);
        vertshuset.hovmester.plasserVedBord(ole,bord);
        vertshuset.hovmester.plasserVedBord(pamula,bord);
        vertshuset.hovmester.plasserVedBord(erik,bord);
        vertshuset.hovmester.plasserVedBord(elise,bord);
        per.velgFraMeny(1, 1);
        ole.velgFraMeny(2, 2);
        pamula.velgFraMeny(1, 1);
        erik.velgFraMeny(1, 1);
        elise.velgFraMeny(1, 1);
        vertshuset.bord4.servitor.taBestilling(per);
        vertshuset.bord4.servitor.taBestilling(ole);
        vertshuset.bord4.servitor.taBestilling(pamula);
        vertshuset.bord4.servitor.taBestilling(erik);
        vertshuset.bord4.servitor.taBestilling(elise);
        vertshuset.kokk.lageMat();
        verthuset.bord4.servitor.hentHovedrett();
        vershuset.bord4.servitor.hentDessert();


              }
}


class Resturant {

    Kelner hovmester;
        Kokk kokk;
    Meny meny;
    Bord bord1;
    Bord bord2;
    Bord bord3;
    Bord bord4;
    Resturant() {
        hovmester = new Kelner("Eva", "dame");
        kokk = new Kokk("Roly", "mann");
```

```
        meny = new Meny();
        bord1 = new Bord(4);
        bord2 = new Bord(2);
        bord3 = new Bord(5);
        bord4 = new Bord(5);
    }

}
```

```
import java.util.Random;



// DENNE MÅ DE STRENGT TATT IKKE SE...
class MollaMat {
        static Thread t = new Thread();
        static void vent(int msek) {
                try {

                        t.sleep(msek);
                } catch(Exception e) {
                }
        }
    public static void main(String [] args) {
        t.start();
        new Kontroll();

    }
}

// DENNE MÅ DE STRENGT TATT IKKE SE...
// MÅ LAGE EN API FOR DEN. IKKE BESKRIV NOEN METODER.
class Person {
    String navn;
    String kjønn;

    Person(String navn, String kjønn) {
        this.navn = navn;
        this.kjønn = kjønn;
    }

    void velgFraMeny(int hoved, int dessert) {}
    int beOmHovedrett() { return 0; }
    int beOmDessert() { return 0; }
    void leggInnBestilling(int bestillt) {}
    void lageMat() {}
    Matrett leverHovedrett() { return null; }
    Matrett leverDessert() { return null; }
    void hentHovedrett() {}
    void hentDessert() {}
    void leverMatrett(Person gjest) {}
    void taBestilling(Person gjest) {}
    void plasserVedBord(Person gjest, Bord bord) {}
    void endreAntallPlasser(Bord bord, int antall) {}
    int hovedrett() { return 0; }
    int dessert() { return 0; }

}

// DENNE MÅ DET LAGES API FOR.
class Gjest extends Person {

    int hovedrett;
    int dessert;
    Meny meny;

    Gjest(String navn, String kjønn) {
        super(navn, kjønn);
        meny = new Meny();
    }
```

```
    // DISSE MÅ IKKE LAGES API FOR
    int hovedrett() {return hovedrett;}
    int dessert() {return dessert;}

    // DENNE METODEN MÅ DET LAGES API FOR:
    // VELGER DESSERT OG HOVEDRETT FRA MENYEN
    // SLIK AT GJESTEN KAN FORTELLE DETTE TIL
    // SERVITØREN NÅR HAN KOMMER.
    void velgFraMeny(int hoved, int dessert) {
        if(meny.meny[hoved] instanceof Dessert) {
            System.out.println("Kan ikke bestille dessert til hovedrett!");
            System.exit(0);
        }
        if(meny.meny[dessert] instanceof Hovedrett) {
            System.out.println("Kan ikke bestille hovedrett til dessert!");
            System.exit(0);
        }
        this.hovedrett = hoved;
        this.dessert = dessert;
    }


    // TRENGER IKKE API, ELLER?
    int beOmHovedrett() {
        return hovedrett;
    }

    // TRENGER IKKE API, ELLER?
    int beOmDessert() {
        return dessert;
    }

}

// DENNE MÅ DET LAGES API FOR
class Kokk extends Person {

    Random r = new Random();
    static int [] bestillinger;
    static boolean lagetMat;
    Meny meny;

    Kokk(String navn, String kjønn) {
        super(navn, kjønn);
        bestillinger = new int[20];
        for(int i = 0; i < bestillinger.length; i++) {
            bestillinger[i] = −1;
        }
        lagetMat = false;
        meny = new Meny();
    }

    // TRENGER IKKE API, ELLER?
    void leggInnBestilling(int bestillt) {
        int plass = r.nextInt(19);

        while(bestillinger[plass] != −1) { plass = r.nextInt(19); }

        bestillinger[plass] = bestillt;
        System.out.println(meny.meny[bestillt].rett + " er bestillt.");
        MollaMat.vent(1000);
    }
    // MÅ LAGE API FOR
    // MÅ KALLES FØR MATEN KAN LEVERES.
    void lageMat() {
        System.out.print("Kokken lager mat...");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.print(".");
        MollaMat.vent(500);
        System.out.println(".ferdig!");
        MollaMat.vent(500);
        lagetMat = true;
    }

    // TRENGER IKKE API, ELLER?
    // SENDER HOVEDRETT TIL KELNER
    Matrett leverHovedrett() {
        if(lagetMat == false) {
            System.out.println("Maten er ikke laget!");
            System.exit(0);
        }

        for(int i = 0; i < bestillinger.length; i++) {
            if(bestillinger[i] != -1 && meny.meny[bestillinger[i]] instanceof Ho
vedrett) {
                int temp = bestillinger[i];
                bestillinger[i] = -1;
                return meny.meny[temp];
            }
        }
        System.out.println("Ukjent matrett.");
        return null;
    }

    // TRENGER IKKE API, ELLER?
    // SENDER DESSERT TIL KELNER
    Matrett leverDessert() {
        if(lagetMat == false) {
            System.out.println("Maten er ikke laget!");
            System.exit(0);
        }
        for(int i = 0; i < bestillinger.length; i++) {
            if(bestillinger[i] != -1 && meny.meny[bestillinger[i]] instanceof De
ssert) {
                int temp = bestillinger[i];
                bestillinger[i] = -1;
                return meny.meny[temp];
            }
        }
        System.out.println("Ukjent matrett.");
        return null;
    }
```

```
}


class Kelner extends Person {

    Matrett ferdigMatrett;
    Kokk kokk;
    Meny meny;

    Kelner(String navn, String kjønn) {
        super(navn, kjønn);
        kokk = new Kokk("Olav", "Mann");
        meny = new Meny();
    }

    // MÅ LAGE API FOR
    // FINNER RIKTIG GJEST SOM SKAL HA MATEN
    // RETURNERER DENNE GJESTEN
    Person finnRiktigGjest(Bord bord) {
        int matrett = meny.rettNummer(ferdigMatrett.rett);

        for(int i = 0; i < bord.antallPlasser; i++) {
            if(ferdigMatrett instanceof Hovedrett) {
                if(matrett == bord.gjester[i].hovedrett()) {
                    return bord.gjester[i];
                }
            } else {
                if(matrett == bord.gjester[i].dessert()) {
                    return bord.gjester[i];
                }
            }
        }

        return null;
    }


    // MÅ LAGE API FOR
    // HENTER EN FERDIG LAGET HOVEDRETT FRA KOKKEN
    // DENNE KOMMER UT I RANDOM!
    void hentHovedrett() {
        // legg inn random rett fra bestillingslisten
        // alle hovedretter kommer før desserter
        ferdigMatrett = kokk.leverHovedrett();
    }

    // MÅ LAGE API FOR
    // HENTER EN FERDIG LAGET DESSERT FRA KOKKEN
    // DENNE KOMMER UT I RANDOM!
    void hentDessert() {
        ferdigMatrett = kokk.leverDessert();
    }

    // MÅ LAGE API FOR
    // LEVERER MATRETT TIL EN GJEST
    void leverMatrett(Person gjest) {
        String type;
        if(ferdigMatrett instanceof Hovedrett)
            type = "hovedrett.";
        else
            type = "dessert.";
        System.out.print(gjest.navn + " har fått " + ferdigMatrett.rett + " til "+ type
);
        if(type.equals("hovedrett.") && ferdigMatrett.rett.equals(meny.meny[gjest.h
ovedrett()].rett) ||
           type.equals("dessert.") && ferdigMatrett.rett.equals(meny.meny[gjest.de
ssert()].rett)) {
                        System.out.println(" Og dette er riktig.");
           } else {
                        System.out.print(" Men dette er feil, ");
                        System.out.print(gjest.navn + " bestillte ");
                        if(type.equals("hovedrett.")) {
                                System.out.println(meny.meny[gjest.hovedrett()].
rett);
                        } else {
                                System.out.println(meny.meny[gjest.dessert()].re
tt);
                        }
           }
           MollaMat.vent(1000);
    }

    // MÅ LAGE API FOR
    // TAR BESTILLING FRA EN GJEST
    void taBestilling(Person gjest) {
        // Legg inn bestilling
        kokk.leggInnBestilling(gjest.hovedrett());
        kokk.leggInnBestilling(gjest.dessert());
    }

    // MÅ LAGE API FOR
    // PLASSERER GJESTER VED ET BORD
    void plasserVedBord(Person gjest, Bord bord) {
        boolean full = bord.leggTilGjest(gjest);

        if(full) {
            System.out.println("Det er ikke plass!");
            System.exit(0);
        }
        System.out.println(gjest.navn + " er lagt til som gjest.");
        MollaMat.vent(1500);
    }

    // MÅ LAGE API FOR
    // ENDRER ANTALL PLASSER VED ET BORD
    void endreAntallPlasser(Bord bord, int antall) {
        bord.nyttAntallPlasser(antall);
    }

}

// DENNE MÅ DE STRENGT TATT IKKE SE...
class Matrett {
    String rett;

    Matrett(String rett) {
        this.rett = rett;
    }
}

// DENNE MÅ DE STRENGT TATT IKKE SE...
class Hovedrett extends Matrett {
    Hovedrett(String rett) {
```

```
        super(rett);
    }
}

// DENNE MÅ DE STRENGT TATT IKKE SE...
class Dessert extends Matrett {
    Dessert(String rett) {
        super(rett);
    }
}

// MÅ LAGE API FOR
// SKRIV OPP ALLE MATRETTER I RIKTIG REKKEFØLGE
class Meny {
    Matrett [] meny;

    Meny() {
        meny = new Matrett[12];
        meny[0] = new Hovedrett("Biff");
        meny[1] = new Hovedrett("Elgstek");
        meny[2] = new Hovedrett("Salat");
        meny[3] = new Hovedrett("Laks");
        meny[4] = new Hovedrett("Kotelett");
        meny[5] = new Hovedrett("Oksestek");
        meny[6] = new Dessert("Is");
        meny[7] = new Dessert("Kake");
        meny[8] = new Dessert("Bær");
        meny[9] = new Dessert("Sjokoladepudding");
        meny[10] = new Dessert("Gele");
        meny[11] = new Dessert("Tiramisu");
    }

    // MÅ LAGE API FOR
    // RETURNERER NUMMERET PÅ MATRETTEN
    int rettNummer(String matrett) {
        int i = 0;

        try {
            while(!matrett.equalsIgnoreCase(meny[i].rett)){
                i++;
            }
        } catch(Exception e) {
            System.out.println("Matrett " + matrett + " finnes ikke.");
            System.exit(0);
        }

        return i;

    }
}

// DENNE MÅ DE STRENGT TATT IKKE SE...
class Bord {

        boolean juksefull;
    int antallPlasser;
    Person [] gjester;
    Person gjest1, gjest2, gjest3, gjest4, gjest5, gjest6;
    Kelner servitør;

    Bord(int plasser) {
        antallPlasser = plasser;
        gjester = new Person[6];
        servitør = new Kelner("Kåre", "Mann");
    }

    void nyttAntallPlasser(int antall) {
        antallPlasser = antall;
        gjester = new Gjest[6];
    }

    boolean leggTilGjest(Person gjest) {



        boolean full = true;

        for(int i = 0; i < antallPlasser; i++) {
            if(gjester[i] == null) {
                gjester[i] = gjest;
                full = false;
                break;
            }
        }

        gjest1 = gjester[0];
        gjest2 = gjester[1];
        gjest3 = gjester[2];
        gjest4 = gjester[3];
        gjest5 = gjester[4];
        gjest6 = gjester[5];

        return full;
    }



}

class Display {

        Display() {
        }

        void skriv(String text) {
                System.out.println(text);
        }

        void skriv(int tall) {
                System.out.println(tall);
        }
}
```

Gruppe 1. Christian som hjelper jentene.
De har et ganske stort og komplisert program. Christian hjelper dem å kommentere ut en bolk tekst.
220: De lurer på om programmet gjorde som det skulle. De synes også at bilen snudde litt sakte. De har også laget en metode snu som de kaller ifra kontrollklassen.
260: De sletter så kommandoene i kontrollklassen om å starte med å kjøre begge motorer forover, og kommenterer så inn igjen den utkommenterte teksten. De har fortsatt et langt program som ikke synes å gjøre noe spesielt.
325: De kommenterers å ut øverste bolk av programmet. Programmet har først kommandoer for å kjøre motorer fremover litt og så stoppe, og så går det inn i en while-løkke som får motorene til å kjøre fremover så lenge ikke lyssensoren ser noe svart.
340: De rekker nå opp hendene og spør Richard hvordan de kan få roboten til å skrive navn.. (på displayet)
400: dhar fra før av laget speakeren i kontroll-klassen. Dette er jo feil, selv om det fungerer. Richard sier nå at de kan legge display på samme sted.
500: Richard prøver å få elevene til å finne ut hvordan de skal bruke Display og skriv-metoden
525. De starter med å skrive skriv.#noe# men siden de husker at det skal være et punktum et sted, så kommer de på at de må skrive beate.skriv Dette sier de er fordi det er displayet som skal skrive ut.
575: De har fortsatt ikke deklarert displayet, bare sagt beate = new Display. Richard spør hva som mangler. De finner ikke ut av dette selv, selv med mye hjelp.
810: Roboten gjorde tydeligvis ikke som den skulle, den skrev ikke det de sa den skulle. Dette er sannsynligvis fordi de ikke venter noen etter de sier at den skal skrive "hallo"

Ikke mye nyttig informasjon i denne snutten.

Gruppe 2. Programmering av restaurant. Hjulpet av Christian.

30: Jenta på gruppa spør om "man skal skrive et mat her?"
De sier de ikke kjønner hva de skal gjøre.
60: Hvorfor skriver de vertshuset.hovmester?   Fordi det er hovmesteren på vertshuset.
Siden det også er vertshuset.bord2 så deduserer jentene at da skal det være vertshuset.meny

85: Det er ikke så logisk at man må skrive vertshuset.hovmester.endreAntallPlasser(vertshuset.bord2) istedenfor vertshuset.hovmester.endreAntallPlasser(bord2)

120: Jentene vil tenke minst mulig selv. De spør Christian før de prøver å tenke logisk selv. Det er også litt forvirring i hvilke gjester som er hvilke nummer.

160: Nok et eksempel på avsporing pga. kameraer som filmer. Selv om nå halvparten av gruppa ønsker å jobbe videre.

170: Selv om de synes det var logisk at man måtte skrive vertshuset.bord2 så gjorde de likevel med neste metode plasserVedBord ikke noe forsøk på å fortelle hvilket bort cecilie skulle plasseres ved.

220: snakker om å lage en historie ved siden av. Skrive med grønt.. er kommentar = historie?

290: Spørsmål ifra sidekamerat: Hvordan laget dere skjerm? new Display...

De hadde så 7 feil under kompilering. Det synes de var mye.

Feilen er bl.a at de ikke har fortalt hvor bord2 ligger, og at de skrev vertshuset.kelner istedenfor vertshuset.hovmester. Det er tydelig at de har tenkt litt selv iom de skrev kelner. Jeg har tro på at de finner ut av dette selv.

350: Jenta sier etter å ha kikket på koden: "vi må få mye hjelp..jeg skjønner ingenting." I tilleg så tar de vekk parameterene til cecilie.valgFraMeny.

370: De tror at feilen er pga noen whitespaces. Det har de ikke rett i.

390: Når de fjernet whitespace og paramererene til valgFraMeny så sa kompilatoren at de bare hadde 1 feil. Dette er trist at den juger på denne måten.

Det er iallefall sikkert at jentene ikke gjør noe forsøk på å forstå programmet, bare prøver å fjerne kompileringsfeil. Dette er et typisk eksempel på: (patching uten forståelse)

405: De tar så vekk den linjen som de får feil på og sier at Christian skal ordne ting for dem.

450: Ole kom bort til gjengen å spør om de står fast.

480: Den ene jenta utbryter at hun vet feilen...at bord skal skrives med stor B. Dette tyder på at de ikke har forstått poenget med stor b noen steder, men de har iallefall fått meg seg case-sensivitet. Men makkeren er ikke enig i at det er feilen.

520: Stor b var ikke løsningen. Likevel fortsetter de å rette små b-er til store b-er.

590: Christian spør hvorfor de ikke bruker bord4, og de svarer at de fjernet bord4. Det tyder på at de iallefall ikke ser på bordene som statiske, men som noe de selv kan putte inn i restauranten, og fjerne som de vil.

Det starter med at de gjør om alle hovmesterene til kelnere, av ukjent grunn. Idet de kompilerer så får de feil både på kelner og hovmester.

55: De legger til en kelner i restauranten, men da spør den andre på gruppa hvorfor de da ikke bare gjør om hovmester til kelner.. (regner med det er navnene de mener)

115: For å løse eventuell kelner-feil så gjør de om alle kelner til kelner1 . Det ser ikke ut som om det er noe forskjell på kelner og hovmester i bruksområder.

130: Videre gjør de om de resterende hovmesterne til kelnere.

160: De kompilerer på nytt og får 5 feil. Den klager på at den ikke finner metoden leverMatrett i Kelner.

175: De sier at det er noe galt med kelneren, men uten å diskutere det sammen rekker de opp handa å spør etter hjelp. Mens hjelpen er på vei, finner de ut at de har skrevet Kelner med liten k i deklarasjonen. Jeg tror dette er forholdsvis smarte jenter, men fy faen så fjortisser de er.

** 190: Hun ene på gruppa sier at siden de hadde laget kelnern på feil måte, så vil stedene kelnern blir brukt, også bli feil. Dette er veldig bra observert. Det triste er at det ikke hjalp i dette tilfellet, siden feilen er noe annet.

225: Hun minst smarte på gruppa lurer på om feilene kan skyldes at de ikke avslutter setningen med 2 parantes slutt, som er gjort endel andre steder. Dette er 100% gjetning og avskrift. Veldig kjedelig. Hun er ikke sikker, men desperasjonen fører til denne typen prøving og feiling.

300: Ole prøver å finne feilen sammen med jentene. Søket etter feilen er bra utført.

Filmen tar slutt før de finner frem til feilen. Feilmeldingen er:

```
cannot resolve symbol
method leverMatrett(String)
kelner1.leverMatrett("henrik")
```

2 grunner.

Enten så er parameteren til leverMatrett feil, eller så finnes ikke leverMatrett i Kelner. Jeg vil vel tro at parameteren til leverMatrett også skulle innehold bordet den skulle blitt levert til.

Hadde elevene vært flinkere på å tolke feilmeldinger, så hadde de nok funnet ut av dette problemet.

Gruppe 3 holder på med restauranten.

Starter med en diskusjon om hva som skal være med under opprettelsen av en person. Jenta vil ikke ha det med, mens gutten sier at det står i api-en at det skal være med. Han er den første jeg har vært innom som bruker api-en.

60: Det blir skiftet til riktig skjerm.

80: Jenta sier at hun føler at de er ute å kjører. Det tyder på at de vil prøve å forstå det de holder på med, og ikke bare prøve og feile. De fortsetter å lage nye gjester. gjest1 til gjest4

130: De snakket om det skulle stå dame eller jente når de deklarerer gjester. De kommer frem til å at det skal stå dame. Jenta klager på at de skulle fått beskjed om det av oss.

180: De spør så om hjelp og mener de er ute å kjører. De har deklarert personene med navn som lise og kjell, mens når de lager nye, så kaller de dem gjest1 - gjest4.

220: Richard forklarer at det må være samsvar mellom per og per.

235: Hva skal man begynne med? Ende sitteplasser ved bordet. De bruker iallefall bruksanvisningen. Det virker som det er lettere å bruke api-en hvis de har oppgavene på samme ark.

gjestene ligger i kontrollklassen og ikke i restauranten. Men dette er vel egentlig riktig.

260: ved å fortelle de at de skulle bruker vertshuset sin hovmester fant de selv frem til hvilken kelner-metode de måtte kalle for å ende antall sitteplasser ved et bord.

280: De gjetter først på at parameteren skal være (bord4, 4) og forteller da ikke at bordet ligger i restauranten. Dette er en klassisk startfeil. Men når Richard spurte hvor bordet tillhørte, så svarte de riktig at det tilhører vertshuset.

320: Gutten lurer på om de må skrive at jestene skal komme, mens jenta tror at de liksom bare er der, noe som forsåvidt er riktig.

360: De synes det er litt tungvindt å måtte plassere alle gjestene ved det samme bordet i flere linjer.. dvs: plasser(per, bord4) plasser(ole, bord4). De vil heller skrive alle personene som skal plasseres ved bordet samtidig..slik: plasser(ole, kjell, kaare, bord4) Men de tror ikke at dette fungerer. Men det er jo egentlig ingen grunn til at det ikke skal fungerere.

390: Når gjestene skal velge ifra menyen synes de dette også er tungvindt å gjøre hver for seg. Men nå vil jeg si at det er mere logisk at dette blir gjort hver for seg.

De lurer på hvordan de skal få tak i per når han skal bestille. Om de skal skrive vertshuset.per   bare per   eller Person per. Det er spørsmål som viser at de tenker iallefall. De antar at det bare skal være per. per.velgFraMeny(2,6)

525: Det er noe de ikke forstår med å "sende tilbake hovedretten som gjesten har bestilt" Dette hopper de da over sålenge. De tror de har gjort mange feil, men de vet at de finner ut av det litt seinere.

580: Over til å ta bestillinger. Dette ser ikke ut til å være noe problem. vertshuset.hovmester tar bestilling i tur og orden for de forskjellige gjestene. Det virker som det er logisk at dette må gjøres på denne måten.

645: Over til kokken tor. De bruker api-en flittig. De regner med at tor ligger i vertshuset og kaller da tor.lagMat. Det viser seg etterhvert at kokken ikke heter tor, men kokk. de forandrer det til kokk.

745: De kommer til å tenke på at hovmesteren har noen oppgaver mens kelneren har andre. Jenta vet at de egentlig kan gjøre akkurat det samme så hun vil ikke at de skal lage noen kelner, mens gutten synes de skal gjøre det siden de har fått beskjed om det.

760: De klarer å lage en ny kelner uten problemer. De gjør så om slik at kelneren tar bestilingene istedenfor hovmesteren.

860: så får de kelneren til å hente hovedrett.

og da er det slutt på del 1.

Dette er en fortsettelse av 3.1 men nå har de kommet endel lenger.

De har noen problemer. Programmet har blitt ganske langt med mye henting av Hovedrett og Dessert.

De setter opp kelneren til å hent hovedrett og levere mat rett etter hverandre. Når de kjører programmet så får de beskjed om at det ikke er noe mat å levere.

Etter at de gjorde noen forandringer så fikk de programmet til å fungere.

Det var veldig lite nyttig å hente ut ifra denne lille videosnutten.

tallet angir antall sekunder ut i filmen

Christian forklarer at variable både må deklareres og initialiseres. Dette er jo helt klart ikke en selvfølge, og er også noe som egentlig gjør java tungvint til denne oppgaven. Det samme gjelder det at det må være stor forbokstav i klassenavn under deklarering.

50: I dette tilfellet så er Lyssensorene deklarert, men ikke initialisert, mens speakeren kun er initialisert men ikke deklarert. Er dette tilfeldig, eller er det en logisk forklaring?

De bytter så ut sine eksisterende trykksensorer med lyssensorer. De sletter initialiseringslinjene for Trykksensorene ivar og iver, og erstatter de med lyssensorene mummi og my.

90: Hun ene på gruppa ville ikke at lyssensoren skulle kobles til samme port som trykksensoren var på, men en annen på gruppa inisterte på at denne også måtte kobles til 3.

175: Siden det kom en fra en annen gruppe å begynte å snakke med jentene, så går gutten alene å prøver roboten. jentene kommer etter, men virker ikke like interessert som gutten. Er det somregel tilfelle at Guttene er mye mer interessert i å prøve, mens jentene bare ser på dette som en vilkensomhelst oppgave som bare må løses, og egentlig ikke er så veldig interessert.

250: Ting gikk tydeligvis ikke helt som først antatt. Ole kommer bort sammen med gutten. Den ene jenta sier til den andre jenta at de "må følge med på Lego" Denne måten å uttrykke seg på viser ikke kjempeinteresse for prosjektet, annet enn at det er en skoleoppgave som må gjøres.

Problemet var lysstyrken sensorene var satt til. De var først satt til 20. Etter forslag ifra Ole, så blir disse satt til 40.

Programmet de lager er bare et lite testprogram som får roboten til å stoppe hvis my-sensoren ser en svart strek. Dette er et godt utgangspunkt.

355: Gutten prøvde roboten såvidt i handa, og nå stoppet den iallefall. De går så bort på gulvet for å prøve ordentlig.

385: Det fungerte fortsatt ikke. Nå blir den ene jenta litt mere med. De vil nå prøve med verdien 50. Det spørs om de finner den riktige verdien på denne måten.

430: Ole forklarer at de kan ha 2 if-tester inni while-løkka slik at de får testet på hver sin lysssensor.

De samarbeider litt bedre når det er 2 av de...jente og gutt. Da er jenta også med på arbeidet. Grupper på 3 har lett for å bli for mye forstyrrelser.

560: De bytter nå til 45. Gutten synes det er rart at bilen stopper når han legger hånda foran sensoren, men ikke ellers. Det er litt rart med slike lyssensorer. De er litt for ømfintelige ovenfor forstyrrelser.

670: gutten kommer alene tilbake og roper etter den ene jenta (hun som jobber litt). Det blir tydeligere og tydeligere at hun egentlig ikke er så veldig interessert.

710: Gutten skriver nå videre på den neste if-løkka. Dette før de har klart å finne riktig lysstyrke å se etter. Når han skriver så sier han punktum imellom volvo.mummi.lysstyrke. Dette tyder på at han ikke tenker på dette som objekter, men heller som noe som bare må skrives.

780: Det skjer noe rundt her. Han sier at "det står for langt bort" ikke sikker på hva han mener med dette. Han har ikke satt klammeparantes for å starte if-klausen, og styrer mye med å få plassert kursoren ett eller annet sted. Jenta sitter passiv i bakgrunnen.

805: Nye beviser kommer frem.. Det viser seg at 45 var det riktige tallet å lete etter. Var dette flaks, eller er dette en god måte å finne frem til tallet man er ute etter på.

klammeparantesen kommer på plass der den skal. Problemet var å finne hvor på tastaturet denne var. Det er ikke så rart. så kopieres det som står i den helt like if-klausen over. Dette også men noen hjelpekommentarer ifra jenta.

890: Ole kommer bort å ber de gjøre om programmet slik at når den ene lyssensoren kommer borti noe svart så svinger bilen, istedenfor å stoppe. Nå skjer det samme på begge sensorene, bilen stopper. Gutten svarer at de må få testet begge sensorene først, før de vil gå noe videre. Jeg synes det er en god strategi.

910: Hvem er hvem av my og mummi? Gutten kan svare på dette, og peker på Roboten. Ole prøver å hinte til hva som må gjøres nå for å få den til å svinge, men gutten insisterer på å få den til å kjøre først.

940: Ole tar over kontrollen og begynner å kommentere bort ting. gutten sier nei, men jenta sier at Ole sikkert vet hva han gjør. Det virker ikke som gutten nå vet hva som vil skje, etter at ole kun stoppet venstremotor ved den ene sensoren og høyre ved den andre. Muligens svinger den litt, men siden den har belter, så blir det nok mye friksjon.

1000: Gutten overfører og går bort for å prøve alene. Det har kommet en ny jente bort for å forstyrre jentene på gruppa. De sitter nå å koser i bakgrunnen.

1100: Gutter kommer tilbake og jentene spør hvordan det gikk. Han svarer ikke, men gjor om ventingen i 3 sekunder om til 2 sekunder. tydeligivs svingte den litt for mye. Richard kommer bort å spør hvordan det går, og gutten sier at de har fått litt hjelp av Kameramannen Ole :)

1270: Hører hoiing i bakgrunnen, så det er tydelig at det fungerer nå.

1345: Gutten og jenta kommer tilbake igjen å gjør om lys-verdien ifra 45 til 43. De gjør også om ventetiden ifra 2 sekunder til 1.5 sekunder. Ventetiden tilsvarer hvor lenge den skal svinge før den begynner å kjøre fremover igjen. Den svinger fortsatt ved å stoppe den ene motoren, ikke ved å kjøre motorene motsatt vei.

1535: Jentene stikker av. Gutten blir sittende alene. Har rydder litt sammen. kanskje det er slutt for dagen? Det kan hende det er lunsjtid.

Christian hjelper gruppe 4. Alle har fått utdelt årbøker. Ikke veldig positivt det heller. I følge koden har de ikke kommet noe videre siden gr_4_opt_1

285: de har problemer med at den ene siden går fortere enn den andre, selv om de i koden har samme fart på begge motorer. Richard ser på problemet. Gutten kikker opp og ned på programmet for å prøve å finne grunnen.
330: Gutten bytter om på portene til lyssensorene i programmet. Han kompilerer å overfører så programmet på nytt. Hva vil han oppnå med dette? han bytter også fysisk om på roboten. Richard spør hvorfor. Noen som sa han skulle prøve det?

370: Resultatet er at den samme fortsatt går mye fortere. Grunnen til at han prøvde å bytte var for å utelukke at problemet lå i programmet eller i RCXen (egentlig samme sak) Dette tyder på at det kanskje er noe som bremser det ene beltet.

420: Richard tar av beltene for å se om de fortsatt går i ulik hastighet. nå så det ut som de gikk like fort. Richard synes motorene virker helt annerledes ifra hverandre. Jeg tror ikke det er tilfelle, det bare virker sånn.

500: Gutten fortsetter å jobbe sammen med Richard. De to jentene tuller i bakgrunnen uten å bidra. Rickard bytter så ut den ene av motorene (fysisk)

590: Bilen så ikke ut til å gå så mye rettere nå.

680: Endelig tilbake til programmet. Gutten mener at den de har laget ikke er så avansert som den som ble laget av oss i teamet. Den var mere "perfekt"

725: De vil at Richard skal få bilen dems til å kjøre når det ikke er sort på noen av sensorene. Det er jo forsåvidt det den allerede gjør, men kanskje de mener at den skal stoppe å svinge når det ikke er sort lenger.

765: Richard gjør om venting i svinging til 100. Dette er et tiendels sekund, og vil sannsynligvis være altfor kort for at maskinen skal klare å følge den svarte streken.

790: Richard lager et Display på roboten. Ber gruppa finne frem API-en. Men Richard skriver alt som skal skrives for å få igang Displayet selv, uten at elevene får prøvd seg på dette.

880: Det ser ut som koden er litt feil, men den kompileres likevel. Det er fordi Richard glemte å lagre før han kompilerte.

955: Vi prøver igjen sier Richard.. Og denne gangen lagrer han først :) Nå gav kompilatoren en feilmelding, og Richard fikset problemet.

1000: Enda et eksempel på at kamera og mikrofon er et distraksjonselement. Kanskje vi skulle vært nøyere på regler i før undersøkelsen begynte.

1070: Richard spør hva som står på displayet, og jenta gjetter at det er hvor mange ganger "den" går rundt. Da kan hun ikke ha fulgt med de siste 10 minuttene.

1150. Gutten tar med seg bilen for å finne ut hva verdien er på gulvet, og på det sorte. Det er bra. Det viser seg at bakken er 45 og tapen er 41. Ikke mye å gå på, med andre ord.

1270: Nå så det ut til at oppdagingen av den svarte tapen fungerte bra, men 100 i svingemillisekunder er altfor lite. Prøver med 200. Gutten er veldig ivrig, mens jenta holder tunga rett i munnen og passer på at alle venterne blir oppdatert.

1435: Gutten overfører programmet 2 ganger "for å være på den sikre siden"   Denne typen ustabilitetr gjør at man kanskje burde vurdere andre verktøy.

1510: Over til gruppen rett ovenfor. Ida spør gutten på gruppa hva "skiten" skal gjøre nå. Dette er ei jente med masse Power, tydeligvis. Hennes ord er lov :I

intervju gruppe 2: synne, jeanette, joakim
Synne og Jeanette synes det var morsomt (sier de iallefall) mens Joakim synes det var gøy først, og så ble det kjedelig etterhvert, siden man ikke gjorde noe mer, som å bruke webkamera og sånt. Eller gjøre bilen større.
1:50 Hva fikk dere bilen til å gjøre.. følge etter tape. Fikk lyset til å fungere (det som bare lyser, ikke sensor), og trykksensor, og å spille tone. I tillegg så skrev de ut på displayet hvilken lysstyrke som ble sett. De begynte på rotasjonssensor, men kom ikke så langt med den.
3:45 Roar: vil at de skal forklare hvordan bilen er bygd opp, og hvordan delene henger sammen. Jentene misforstår og tror jeg mener den fysiske bilen. De vil da at joacim skal forklare. Joacim må likevel tegne etterpå.
4:30 Jeg prøver å forklare at jeg vil ha en tegning av programmet, men jeg forklarer det veldig dårlig. Jeg starter så med å tegne en kontroll-klasse og sammenligner det med kontrolltårnet på en flyplass. ¨
5:20 Jentene spør om de skal tegne på samme måten som ble gjort på tavlen, og det er helt riktig.
5:35 Roar: Kan man ha flere roboter? Joakim: ja, men de kan ikke hete det samme (riktig observert) Roar: hvis man hadde 2 roboter? ¨
6:25 Roar: Hvis man skal sende kommandoer til robot1, hva gjør man?
Svaret er veldig usikkert: kontroll.robot1..... det er jo forsåvidt riktig hvis man bare skal ha tak i roboten, og ikke gi den noen kommandoer.
I følge tegningen så har joacim tegnet 2 roboter og festet de til kontroll, men de går til samme variabel i kontroll-klassen.
7:15 Hva består en robot av? 2 motorer som blir festet på riktig måte på tegningen.
7:40 Roar: Hvorfor blir ikke venstre og høyremotor lagret i kontroll? synne: Fordi de hører til roboten. Bra.
8:00 Roar: Hvor skal assistenten? Joacim vil ha den i roboten, mens jentene ikke er enige i det. De har endel problemer med å tegne hvor assistenten skal være. Jeanette vil ikke, og Joacim vet ikke helt. Synne er overraskende mye med i diskusjonene i forhold til hva hun gjorde på gruppa.
9:20 Kontroll sender per ut til begge 2, dvs begge robotene. Dette er også en måte å tenke på det på. At kommandoene blir sendt ut til robotene via via per. Jeg sier at dette er feil. JEg forsto det som de trodde at per måtte ligge i roboten på en måte.
10:35 Roar: Kan man skrive per.kjørfremover? Jeanette: Nei, det er motoren som gjør sånne ting, per er bare en assistent. Det er forståvidt helt riktig det. Vet ikke helt om de har forstått hva som er likheter og forskjeller mellom de likevel.
10:50 Når de skulle skrive hva som var i roboten, skrev de inn metodene kjørfremover og kjørbakover i roboten. Dette stemmer ikke utifra utdelt kode, men det stemmer hvis de har laget slike metoder selv.
De har laget en metode som heter snu().
11:00 De er enige med Roar når han sier at de har feil i at det ikke finnes noen kjørfremover i robot.

11:50 Laging av et hus.
Hus hansen.
12:40 Roar: Fest en sofa til huset. Alle tenker litt, og så kommer Jocim på det, og tegner er firkant-sofa og fester den til huset. Egenskap til sofa: den kan sittes i.
13:20 Jocaim spør: skal vi ha en assistent? Et tegn på at assistenten i roboten forvirrer, siden den egentlig ikke har noe der å gjøre, men må være med for å få programmet til å fungere. Det samme hadde kanskje vært tilfelle med et hus.

13:50 Jocam spør: Hvis man ikke har noen assistent, vil ikke programmet lage sofaer hele tiden? Dette er jo et veldig godt spørsmål. Hvis sofa-simuleringen er på samme måte som robot-programmet så er man avhengig av noe som kan vente.
14:20 Joacim skriver i assistent-klassen - per.vent, selv om det bare skal stå "vent" her. Jeg vil tro utifra dette, at han ikke helt forstår hvorfor vi må skrive per.vent i kontroll-klassen. Det er dumt og forvirrende med kontroll-klassen.
15:20 Lager en vaskemaskin på samme måte.
15:35 Hvordan vil man lage en nabo. Joacim tegner helt riktig et Hus jensen hvor pekeren ligger i Kontroll.

16:00 Her gjør jeg samme feilen som med gruppe 2. Jeg lar de tegne 2 forskjellige hus av typen Hus, som er litt forskjellige, og dette går ikke ann med mindre man lager 2 forskjellige subklasser.

17:00 Roar: Hvis man vil ha 2 senger, hva gjør man da? Synne sammenligner det med Motor, og dette er en veldig bra sammenligning.
18:00 Roar: Hva hvis huset skal ha en gartner, hvor skal den legges? Jeanette: da må vi skrive sånn torleif = new Gartner.. Det er helt riktig.
Synne: Da kan man gjøre det samme med per = new Assistent.
19:00 Joacim tegner den en gartner på arket, og Roar spør om det blir riktig. Kaller den nye klassen for torleif istedenfor Gartner. Det sitter tydeligvis ikke helt hos Joacim forskjell på navn og på type (klasse)
1940: Roar spør om vi har en type som heter Torleif, og de sier nei.
1950: Synne foreslår å ha gartneren i Kontroll sammen med assistenten.
20:20 Synne vil lage en hage. Roar sier at de kan ha garneren i denne hagen. Jeanette vil ha en hage i Kontroll. fordi det blir det samme som med husene..Men Synne mener at Hagen hører til huset. De er usikre, men Synne mener at hvis den står i Kontroll så blir den et eget hus og ikke en hage tilhørende et hus. Det er nok et problem å tenke i størrelse, når en hage blir for stor til å kunne "festes" til et hus. De blir enige om at hagen tilhører huset.
22:00 Putter så en Gartner ved navn torleif i hagen.
Merker også at Roar er inkonsistent på bruk av ordet egenskap, når det blir brukt både om metoder til objekter, og objekter som et objekt peker på.
23:00 Roar vil at garneren skal begynne å renske opp i hagen. Joakom foreslår kontroll.hushansen. hage.gartner.gjørpent(1000)
Her er det mange feil. kontroll kan sees på som this, men han tenker nok at man likevel står utenfor kontroll når man begynner. hushansen blir skrevet fordi det står på arket. Hus var der ment som en deklarasjon av hansen. hage fikk aldri noe navn, så dette er greit, men gartner er terpet så mye på at gartneren heter torleif, at dette burde han ikke gjort feil. Ellers så tenker han riktig når han går inn til garneren.

23:35 Jentene synes at det ble skrevet for mye. De mener at når hagen er en del av huset, så vet programmet at gartneren er i hagen. Det er tydeligvis stor forskjell på hage og sofa i måten de er en del av huset.
24:30: Det er tydelig at Roar ikke gidder å bry seg om han skriver typen eller navnet på variabelen når han dott-er seg frem. Parameteren tenker han helt riktig når han vil gjøre pent i 10000 millisekunder.

katinka, margrete, aleksander
R: Hva var gøy? margrete: prorammering mest, og litt å bygge.
R: Vanskelig? margrete: mye info i starten, gikk bedre etterhvert, begynte å gå litt automatisk.
1:20 Fikk ikke helt til å følge den svarte streken. Bilen kjørte for fort og lysstyrken var ikke helt justert riktig. Programmet fungerte egentlig likevel.
2:00 De lagde en serie av kommandoer. først frem og så tilbake osv. Er det noen forskjell på å lage serie av kommandoer istedenfor noe som oppfører seg etter det som er rundt.
3:00 Hva ble laget i tillegg til robot....motorene kjell og kåre. Hvor ble de laget? På roboten.
3:45 margrete: skal jeg skrive høyremotor og sånt? ... Har navnet høyremotor noe å si for hvor motoren sitter? Hun vet at de kalte den kaare, er den høyremotor? Burde vi her hatt subklasser høyremotor og venstremotor? nei..jeg tror ikke det.
4:10 Hun tegner en firkant Motor og inni skriver hun kjell og kåre. Dette er kanskje en vanlig måte å tenke på. R: Lagde dere en motor som het kjell og kaare? margrete: nei...og så tegner hun en boks til som heter kaare.
4:30 R: Lagde dere noe mer på bilen? M: Assistent.... R: Lagde dere den på bilen? M: Nei...i kontroll. I tillegg 2 lyssensorer på roboten.
5:30 Når hun tegner på lyssensorer så tegner hun også kun 1 boks. Er det logisk at det skal være 2 bokser?
5:45 R: Finn 2 like ting på roboten... m: motorene er like og lyssensorene er like R: Hvorfor? M: Begge har samme egenskaper.
6:20 R: Hva skrives for å få bilen til å kjøre fremover? donny.høyremotor.kjørfremover()
6:55 R: Lagde dere metode i roboten? m: lagde snumetode.
7:30 M: i snu-metoden i roboten skrev vi donny.hm.kjør og donny.hm.bak og så vent. R: Det tror jeg var feil. Veit roboten hvem donny er? M: Vi trengte ikke skrive donny.   Men vet hun hvorfor? Jeg vet ikke helt om hun forstår hva som skjer når hun skriver donny.hm.kjør.
8:10 R: De lagde sannsynligvis en ny assistent i roboten for å få med vent-funksjonen. Dette er jo kjempeforvirrende. Det er veldig, veldig dumt at vi måtte ha med denne assistenten. Bare det i seg selv forteller at lejos kanskje ikke er tingen.
8:20 R: Hva vis dere skal snu?.. hva skriver dere etter donny.hm.kjørfremover? M: men vi skulle jo ikke skrive donny. R: Jo, ifra kontroll skal dere det. (Dette blir veldig forvirrende)
8:40 M: Vi skriver donny.snu()   og det er jo bra at hun trodde.
9:00 Lager en assistent i donny også. Kan kontroll få tak i assistenten til donny...ja....kan donny få tak i assistenten til kontroll....nei.   Supert. Det er helt riktig. (hvorfor forstår hun dette) Hva er det logiske med dette?

9:40 Over til laging av hus..

hus ville.
10:10 legge til vindu. m: må man kalle vinduet noe? R: Ja, må alltid kalle noe noe.
11:00 Forskjellen på dør1 og dør2? M: stedene de er festet.
11:30 Hvis man vil ha en annen type dør? M: Da må man kalle den noe annent.. annet navn på døra? nei annen dørtype.   Det er riktig, iallefall når man ikke tenker på subklasser.
12:00 m: hvis vi skal legge til en hundedør, må vi kalle dør2 hundedør? Litt usikker på hvorfor hun tror at man ikke kan ha 3 dører på et hus. Det kan være fordi man aldri har mer enn 2 ting av en sort på roboten.

12:15 R: Er alle dørene like? Nei, hundedør er ulik. Hva vil det si at de er like? At de har de samme egenskapene.
13:00 Hagen blir plassert utenfor huset. Hvorfor det? m: Fordi hagen ikke sitter på huset, men er utenfor huset, og kan være for seg selv.
Det er jo et godt poeng at man f.eks kan gå inn i hagen uten først å gå inn i huset. Vil da hagen tilhøre huset?
14:00 En park er ikke en del av et hus.
14:20 Så blir en huske plassert i hagen, og så et barn blir plassert på husken. Helt riktig. Barnet får så egenskapen å huske.
15:30 Åpning av dør1. M: Da må det først være en "link" ifra kontroll til ville. Det er kjempebra at hun fikk med seg dette. Det viser forståelse. Ellers ble det riktig med ville.dør.åpne.
17:00 Det dukker opp en morder i kontroll som skal drepe det huskende barnet.
M: morder.lilleputt???? dere må være med å diskutere. M: hva med.. morder.lilleputt.drep.hage1. huske.barn
18:50 R: Dere har en feil i førsten. Katinka gjetter at de skulle hatt med huset.
De finner ikke ut av feilen selv, så Roar hjelper de med å si at det begynte med morder.
Roar må forklare om parantes. Dette var ikke noe gruppen forsto av seg selv. Opplegget forklarer for lite om parametre. Ellers så tenkte gruppa ganske riktig.

Intervju med (fra venstre til høyre) ståle, ida og magnus. (ark2)

35: Har dere lært noe...ja..
Hva?   Å programmere, bygge Lego.
55: Ida: Skjønne litt mer hvordan ting fungerer. f.eks legobiler

65: Kjedelig når man ikke skjønte hva man skulle gjøre og ikke får det til. F.eks oppgaven med Resturanten
Gøy med utfordringer, men ikke når man ikke kommer videre.
85: Magnus syntes at lego-byggingen var gøy.

110: Lagde en bil som het arsenal, det samme som programmet het.

R: Hva bestod bilen av? ståle: motor... Det stod Legomotor som type. Roar forklarer at Legomotor er en type motor, men siden de ikke vet om noen andre typer motorer så har det ikke så mye å si.
150: Ståle: 2 motorer eller 1. R: Hva ble gjort i programmet? S: Høyremotor og venstremotor.
170: R: Så kan dere tegne motorene på samme måte som jeg tegnet Robot. Ståle tegne så den ene motororen, og da Roar spør om de bare hadde 1 motor, så tegner Ida den andre motoren på samme måte.
210: Ståle sier at Roboten også består av RCX-en. Dette er jo forsåvidt sant, men det er ikke noe som modelleres i programmet.
220: Ida sier at den bestod av en lyssensor. Tegner så denne og lager en ny firkant.
250: R: Den bestod også av en trykksensor. Ida: 2 trykksensorer.
Til forskjell ifra legomotorene så blir trykksensoren bare skrevet inn i roboten 1 gang, mens det ble laget 2 streker ut til symboliserende firkanter. Er dette tilfeldig? Det kommer ikke inn i samtalen.
275: : Hva kan dere få bilen til å gjøre? S: kjøre fremover, stoppe, merke svart strek på gulvet. R: Hvordan merket den svart strek? S: Den sjekket lysstyrken.   Dette er vel egentlig svar godt nok. De jobbet mye med å finne riktig verdi å sjekke etter.
300: Snakke om de forskjellige delene til roboten. Er noen deler like? Ida: Begge motorene. R: Er de helt like? De diskuterer litt, nei, jo....De trakk forskjellig?
335: R: Motorene er 2 fysisk forskjellige ting med nøyaktig de samme egenskapene.
340: R: Hva med LegoMotor og Lyssensor? Magnus: Legomotor og Lyssensor har forskjellige egenskaper. Endel diskusjoner.... S: Den ene kjører, går rundt....Ida: Den ene er drivkraften, mens den andre må bruke drivkraften for å gjøre noe.. Hmm. Ida mener hun bare tuller. :)
370: Ida: Motoren er den som lager energi slik at lyssensoren virker??? Heftige diskusjoner. Ståle og Magnus overtaler Ida til at RCxen gir strøm til både motor og sensor.
405: Er RCXen et stort batteri? JA..forsåvidt.
425: R: Hadde dere noe mer enn en robot? Belter og Hjul....slapp å programmere. Musikk. Display.. Hvor var disse? I roboten.. Et ting til.. Ida: En assistent...hvor da...i kontroll.. Dette er riktig.
Ida skriver bob(assistend) i kontroll. Jeg sier at hun må lage assistenten på samme måte som hun lagde andre ting. De tegner så en bob-firkant med 2 metoder...vent og tilfeldigtall.
505: Hva skrev man for å få roboten til å kjøre fremover? De måtte først ha en "greie" inni roboten. Regner med at det er deklarering og initialiserig av en motor. Så sier Ståle helt riktig: arsenal.hm. kjørfremover( ) R: Hvorfor? S: Peker på arket og sier: først så går man inn her, og så inn der, og så inn i motor og gjør det som gjør at den beveger seg (kaller på riktig metode)

545: Hva med venting? Ida: bob.vent(antall millisekunder). hvorfor ikke bilen.vent? Ida veldig usikker, men forstår at det er noe å tenke på. Ståle forklarer veldig usikker, men jeg tror han forklarer det riktig. Roar er iallefall fornøyd.

Det er kjempetydelig at Ståle (og kanskje Magnus) har litt interesse og forståelse for hva som skjer, mens Ida bare kan det hun har lært uten noen viderere forståelse. Ida er typisk skoleflink for å være det. Hun siterer APIen

570: Huset: (ark1)
Kontroll med en variabel ville som peker på et hus-objekt: Hva vil vi at husket skal bestå av? Magnus: 2 dører. Han ville ikke tegne det inn i huset på arket...overlot det til Ida.
625: Ida tegner inn dører. dør1 og dør2. De snakker i bakgrunnen om bruk av sensorer for å automatisere hus. Type Dør.. navn dør1. 2 Totalt identiske dører. bortsett ifra fargen.
680: Hva hvis man vil ha en elektrisk dør i tillegg til vanlige dører. Navn: blå, type: Elektrisk dør.
730: Forskjeller mellom dørene. Den ene er elektrisk og den andre er ikke. Roar sier at egenskapene er forskjellige??
755: Likheter? Begge brukes til å gå igjennom.
770: Lag stue og kjøkken. Ida mener man må lage et rom før man kan lage en stue. Dette er jo kjempe-objekt-orientert tenking. Var det tilfeldig?. Er huset et rom? Kaller rommene rom1 og rom2 (eller stue og kjøkken)
815: R: Er begge rommene av typen rom, eller er det ene av typen kjøkken og det andre av type stue? Ida: Begge er rom.
825: Her tenker Ida mye. Hun blander sammen navnet på typer (klasser) og navnet på selveste klassene. Sier at begge dørene burde hett dør. Rommene blir nå hetende stue og kjøkken istendenfor rom1 og rom2. Etter en forklaring så er de enige.
895: R: Hvis vi vil ha forskjellige rom?` Ida sier at vi får et problem, noe som er helt riktig. De foreslår å legge airconditioning i rommet på samme måte som de la dører til huset.
935: Ståle mener man burde lagt airconditioning i huset, men blir nedstemt av Ida som vil legge det i rommet. De kaller en airconditioning-type for AC og kaller dette objektet for DC.
980: Egenskapene til aircondition-eren   Den kan blåse varmt og kaldt sier Ida. Roar sier at dette kan skrives som parametere, varmt og kaldt. Vet ikke om folka forsto dette.
1010: R: fra kontroll start airconditioningen. Ståle skriver: ville.kjøkken.dc.blåse(varmt) På arket står det ac -> dc av typen ac men dette er nok bare en skrivefeil. De forklarte det riktig.
1060: Magnus får beskjed om å forklare forfor det skal stå på denne måten.. Han forklarer det sånn passe greit. Man går først til den, og så til den og så til den osv.
1080: Det dukker opp en tyv. med egenskapen stele (bruker egenskap om metoder også). Tyven skal stjele airconditionanlegget ifra veggen. Ida prøver: hun prøver med ståle.stjele(ville.kjøkken.ac) her blir også Ac og DC blandet sammen. her skulle det stått dc. Roar legger ikke merke til dette selv. R: Vil tyven stjele både ville og kjøkken i tillegg. Ida sier hun er usikker. De andre virker sikre på at det ikke vil skje. Ståle sier: det er ville sin stue sin airconditioning.

1340: Roar begynner å forklare om subklasser. Etter en forholdsvis kort forklaring så prøvde Ida å forklare hva de kunne gjort, uten å få det ordentlig til. Jeg mener at de ikke helt forso hva jeg forklarte. Men har de forstått subklasser?

Disse jentene synes det var morsomt å programmere roboten, spesielt når de fikk det til.
På tegningen sin har de kun klasser, men ikke navn på klassene slik som noen av de andre hadde.
Roar sier at ikke dette er så farlig.
1:00 R: Hva kalte dere bilen? C: gunnar.. (men gunnar var navnet på programmet, og ikke bilen som het volvo)
1:20 R: Hva gjode dere med bilen? B: løste oppgave 1 og 2..   Typisk eksempel på at jentene liker å løse oppgaver som kan løses, slik at de blir ferdige.
2:40 B: Sier at venstremotor og høyremotor hadde hver sin sensor. Det er rart å si. De var koblet til den gule dingsen. Kanskje RCXen blir sett på som en Motor på denne gruppa.
4:10 R: Er det forskjell på lyssensor og motor? c: Den ene får hjulene til å gå, den andre fanger opp lys. (forskjellige egenskaper)
4:30 R: Er det forskjell på lyssensorene? Jentene tror ikke det er noen forskjeller, men er litt usikre. Det samme gjelder motorene, bortsett fra at de styrer hver sin side av bilen.
5:00 R: Hva mer? C: Toner (speaker) R: Hvordan? C: Det var assistenten, tror jeg? Dette er vel noe som assistenten kunne gjort, men som det ikke er logisk at assistenten gjør. Det kan være fordi det ikke er noe de ser på RCXen, men noe som bare er der...på samme måte som stoppe-mulighetene.
5:30 Diskutere seg imellom...komme inn på Display (som er synlig, men som ikke blir satt på bilen) Roar måtte si Speaker.
Jentene er veldig usikre på seg selv.
6:20 Er det forskjell på lyssensor og display? den ene føler lys og den andre skriver.
6:40 R: Hvor var assistenten? C: Inni der (peker på robot-klassen). Roar spør igjen om den var i robot-delen av programmet? Jentene fortsatt usikre.
7:05 R: Hva skrev dere hvis dere skulle vente? per.vent Hva med når bilen skulle kjøre fremover? høyre/venstremotor.kjørfremover. (dette kan være fordi de ikke riktig husker. Ingen av jentene ser på tegningen sin mens de diskuterer og kommer med forslag. De kommer på etterhvert at de sa "volvo" først.
7:40 per sitter da på en egen del.. (og tegner den som en del av kontrollklassen)
8:10 Her forvirrer Roar med å sammenligne per.vent med volvo.høyremoter. Dette er helt på trynet.
8:25 R: Dere får tak i per på samme måte som med volvo. Hvor må da per ligge? C: I kontroll.
8:40 De skriver så Per som klassetype istedenfor Assistent.
9:10 De husker hvordan man skulle få bilen til å kjøre fremover.
9:50 En litt dårlig forklaring på hvorfor det må stå volvo.høyremotor.kjørfremover

10:30 Laging av hus
11:00 De vil innrede kjøkkenet i huset for å lage et kjøkken. Dette betyr at et kjøkken er ikke et kjøkken før det består av kjøkkenredskaper
La til et kjøkken på riktig måte. Legger også til bad på riktig måte. Legger så til kjøleskap og komfyr i kjøkkenet på riktig måte.
13:00 Mer terping på egenskaper. Kjøleskap og kjøkken har forskjellige egenskaper.
13:35 Likhet mellom mellom komfyr og kjøleskap: De er begge på kjøkkenet.
13:50 Lager et kjøleskap til. Er det forskjell på kjøleskapene? Nei. de er like
14:00 Hvis man vil ha et kjøleskap på 50 liter istedenfor 100 liter, er det da samme type kjøleskap som de andre?
14:50 Interessant diskusjon om kjøleskap. Litt mumlende at det ikke har noe å si om det er på 50 liter eller på 100 liter. Det er likevel kjøleskap. Det er ikke samme fysiske ting selv om det er samme

ting???
15:20 Roar forklarer at to kjøleskap ikke kan være på forskjellig størrelse da de begge er av typen kjøleskap.
16:30 I kjøleskapet er det en melk som kan eksplodere.
16:50 En fyr i kontroll skal få melken til å eksploder. førse forskøk på løsning er kjøleskap.melk.eksploder()   De fikk beskjed om at de stod i kontroll, så dette er feil. Det er kanskje mer naturlig å lage en metode i kjøkkenet som kan eksplodere melka. Neste forsøk var villa.kjøleskap osv   Roar sier at de glemte av kjøkken, og da blir dette også ført til i rekken.
18:00 stue blir lagt til. Med en person som kan gå. Roar: Få denne personen til å gå til kjøkkenet. Kallet skal gjøres ifra kontroll.
19:25 De får tak i personen på riktig måte i stua. Men som parameter har de bare (kjøkken)
Roar: vet denne mannen om kjøkkenet? De svarer at han bor i huset, så det burde han. De forseslår å styre han manuelt bort til kjøkkenet. Det lar seg ikke gjøre. Det er veldig forstålig at de ikke fikk til denne oppgaven, siden de ikke har gjort det i sitt legoprogram. Det viser også at de ikke helt forstår hvordan verden ser ut i kontroll-klassen.
20:30 Etter mye hjelp så blir det riktig. villa.stue.person.gå(villa.kjøkken)