# Object Oriented Programming (OOP)

Programming computers started with *machine code*. The first programmers used zeros and ones to get something done. As you all can understand, machine code was not too easy to use. You had to memorize eight-bit combination’s meanings. And these meanings varied among computers (i.e. processors).

Then came assembler. In assembler, these eight-bit combinations were replaced by 2-4 letter mnemonics. When you wanted to move some data from location A (usually a so-called register) to B you just typed *MOV A, B.* The assembler “language” made it much easier to code programs. Still the problem of the processor existed. The assembler was different among different processors.

Then high-level languages started to pop up. The first programming languages were written in the 1940’s but they did not get too much attention. This was mainly because there were not so many computers and the few ones were very expensive.

In 1954 the first widely used programming language was invented. FORTRAN is a general-purpose high-level programming language that is still used today. In the next few decades, several programming languages were invented. Some of the languages were specialized to some special purpose and some were meant as general purpose. For example, Pascal was implemented for teaching purposes and Prolog for logic programming.

In the beginning almost, all high-level languages were *procedural*. It meant that the programs consisted of procedures (subroutines). In the 1980’s *object-oriented* languages started to gain popularity. Smalltalk was the first OO-language and there were a lot to come: C++, Ada, Eiffel and many more. The OO-languages were not procedural. They grouped data and the actions (subroutines) into a packet, usually called a class.

These languages still had something in common: a program made by these languages had to be recompiled when moved to another platform. In 1995 the Java-language addressed this problem. A java-program can run on any platform with a *virtual machine* developed for that platform.

Lately a new *paradigm* in programming has popped up: functional programming. It is based on mathematical functions avoiding changing state and mutable data.

## Terminology

A class can have data *members* and *methods* (subroutines). Data represents the model, and the methods are means of handling that data.

An *object* is an instance of a class. An object resists in memory and can be manipulated by the methods it contains.

*Encapsulation* means data hiding. In the OO-paradigm encapsulation is a very important aspect. The idea of encapsulation is that the programmer never reveals more of the contents of a class than is absolutely needed. The idea is thus to keep the implementation as hidden as possible. This usually means that the user of the class has no direct access to the data members and only restricted access to the methods.

## Classes

Here is a base declaration of a class:

class ClockTime {

private int hour;

private int minute;

public ClockTime ( int h, int m ) {

hour = h;

minute = m;

System.out.println (“Allocating”);

}

public void setTime ( int h, int m ) {

hour = h;

minute = m;

}

public int getHour () { return hour; }

public int getMinute () { return minute;}

}

And underneath is a code snippet using this class:

ClockTime ct = new ClockTime ( 13, 42 );

System.out.println ( ct.getHour() + ":" + ct.getMinute());

The class name is given after the keyword *class*. In the data and methods declaration parts, we can have private, protected, public and package access members. If no access type is defined, it is by default package access. Here we have two private members: hour and minute of type int. Because the members are private, they cannot be reached outside the class. If you try to reference these variables, you will get a compiler error.

All the methods have public access which means they can be reached from anywhere. In most cases (methods), this is what you want. The constructor is used for initialization, and it is usually very simple. It should only initiate all or some of the data members. The constructor always has the same name as the class has.

The rest of the class contains methods with which the class data can be manipulated.

Example: Classes\ClockTime.java