

Unit 1

Introduction to programming

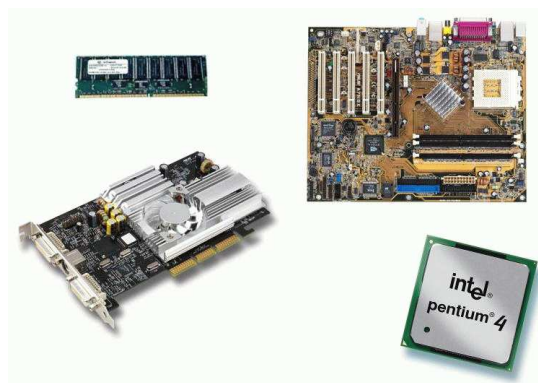
Summary

- Architecture of a computer
- Programming languages
- Program = objects + operations
- First Java program
- Writing, compiling, and executing a program
- Program errors

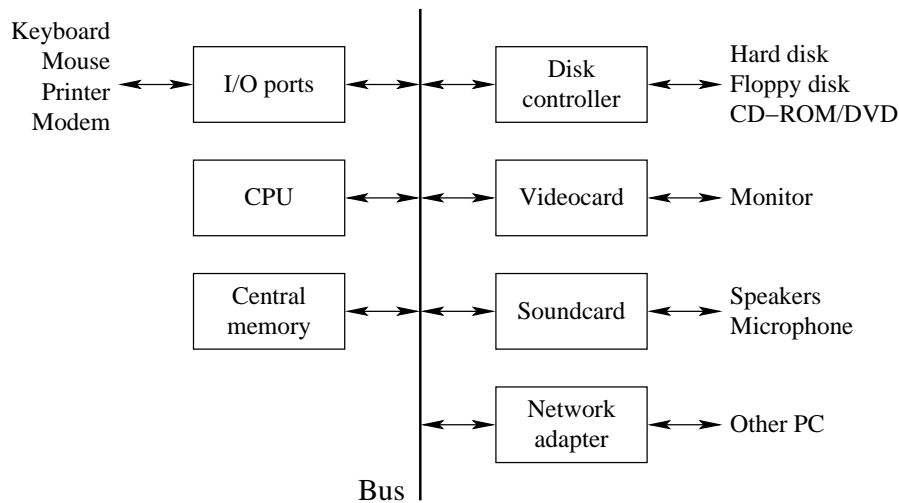
1.1 What is a computer?

- Hardware
 - Processor
 - Memory
 - I/O units
- How does it work?
 - Executes very simple instructions
 - Executes them incredibly fast
 - Must be programmed: it is the software, i.e., the programs, that characterize what a computer actually does

1.2 Components of a computer



1.3 Simplified architecture of a computer



1.4 Languages for programming a computer

- Machine language

```
21 40 16 100 163 240
```

- Assembler language

```
iload intRate
bipush 100
if_icmpgt intError
```

- High level programming languages

```
if (intRate > 100) ...
```

1.5 Programs

The programs characterize what a computer actually does.

A program (independently of the language in which it is written) is constituted by two fundamental parts:

- a representation of the information (data) relative to the domain of interest: **objects**
- a description of how to manipulate the representation in such a way as to realize the desired functionality: **operations**

To write a program both aspects have to be addressed.

1.6 Example: CallCenter

Application: We want to realize a program for a call-center that handles requests for telephone numbers. Specifically, a certain client requests the telephone number associated to a certain person name in a certain city, and the operator answers to the request by selecting the telephone registry of the city and searching there the telephone number corresponding to the person name.

- Objects:
 - client
 - operator
 - telephone registry
 - telephone numbers, person names, cities - these are just character sequences (*strings*)
- Operations:
 1. request for the telephone number of a given person name in a given city, done by a client
 2. selection of the telephone registry of the requested city, done by the operator
 3. search of the telephone number corresponding to the requested person name on the telephone registry

The program has to represent all objects that come into play and realize all operations.

1.7 Representation of the domain: objects

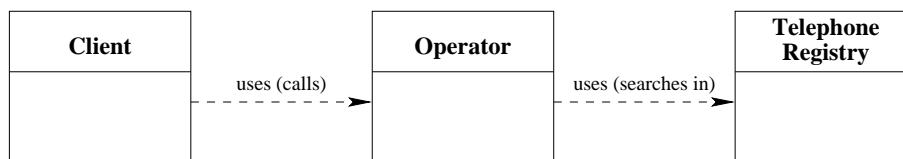
- group objects of the same type into *classes*
- establish the *relations between the classes*, i.e., how the objects of the different classes are connected to each other
- establish the *properties* of the objects belonging to each class
- realize the classes, the relationships between the classes, and the properties of the classes in the selected *programming language*

1.8 Class diagram

To make the classes and the relationships between the classes explicit, we can use the **class diagram**:

- each *class* is denoted by a rectangle containing the class name;
- the *relations* between classes are denoted by arrows; such arrows represent relations between classes in a simplified form, as generic usage relations;
- the properties of classes are not shown.

Example: class diagram of the CallCenter application.



Class diagrams are commonly used in software design. For example, the Unified Modeling Language (UML), which is the de facto standard formalisms for software design, allows one to develop quite sophisticated class diagrams.

1.9 Realization of operations: algorithms

Usually, we realize an operation when we need to solve a specific **problem**.

Example: given a person name, find the corresponding telephone number in a telephone registry.

To delegate to a computer the solution of a problem, it is necessary to find an algorithm that solves the problem.

Algorithm: *procedure through which we obtain the solution of a problem. In other words, a sequence of instructions that, when executed in sequence, allow one to calculate the solution of the problem starting from the information provided as input.*

An algorithm is characterized by:

- **non ambiguity:** the instructions must be interpretable in a unique way by whom is executing them
- **executability:** it must be possible to execute each instruction (in a finite amount of time) given the available resources
- **finiteness:** the execution of the algorithm must terminate in a finite amount of time for each possible set of input data

Example of an algorithm: scan the person names, one after the other as they appear in the registry, until you have found the requested one; return the associated telephone number.

Are there other algorithms to solve the same problem? Yes!

Once we have found/developed an algorithm, we have to code it in the selected programming language.

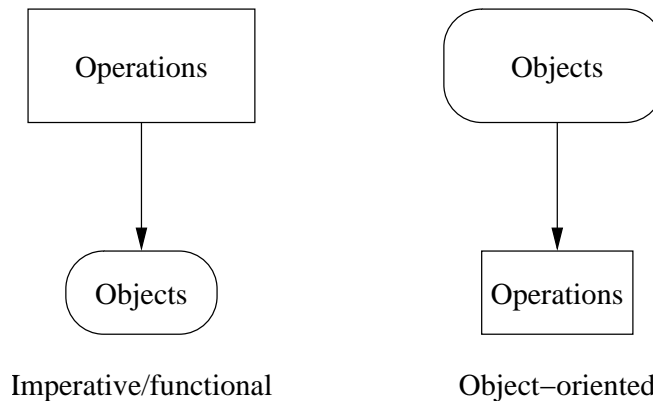
1.10 Programming paradigms

There are several programming paradigms, that differ in the emphasis they put on the two fundamental aspects: objects and operations.

The main programming paradigms are:

1. **imperative:** the emphasis is on the operations intended as actions/commands/instructions that change the state of the computation; the objects are functional to the computation;
2. **functional:** the emphasis is on the operations intended as functions that compute results; the objects are functional to the computation;

3. **object oriented**: the emphasis is on the objects, which as a whole represent the domain of interest; the operations are functional to the representation.



Usually, in a program different programming paradigms are used. Hence, programming languages provide support (with different degrees) for the various paradigms.

1.11 Java

In this course we will use the **Java** programming language.

Java is a *modern, high level, object oriented* programming language, which supports also the *imperative* and the *functional* programming paradigms.

General characteristics of Java:

- simple
- platform independent (the same program can be run on Windows, Unix, MacOS, etc.)
- comes equipped with a very rich set of well developed libraries
- designed for the use on Internet
- based on virtual machine (see later)
- safe (the virtual machine forbids undesired accesses to applications running via the Internet)

1.12 The first Java program

```

import java.lang.*;

public class First {
    public static void main(String[] args) {
        System.out.println("This is my first Java program.");
    }
}
  
```

The statements have the following meaning:

- `import java.lang.*;` request to use libraries of predefined classes/programs (in fact, the `java.lang` library is imported automatically, hence this statement can be omitted)
- `public class First {...}` definition of a class/program called `First`
- `public static void main(String[] args) {...}` definition of the `main` method (a method is the realization of an operation in Java)
- `System.out.println("This is my first Java program.");` statement to print a message on the video
- `System.out` predefined object/instance of the predefined class `PrintStream`
- `println` method of the class `PrintStream` applied to the object `System.out`
- `"This is my first Java program."` object of the class `String` representing the sentence to display

*Note: Java is **case-sensitive**, i.e., there is a difference between lower-case and upper-case letters. E.g., `class` is different from `Class`.*

1.13 Second program

```
public class Second {
    public static void main(String[] args) {
        System.out.println("This is my second Java program ...");
        System.out.println("... and it will not be my last one.");
    }
}
```

The sequence of two statements means that the two statements will be executed in the order in which they appear in the program.

1.14 Write, compile, and execute a Java program

1. preparation of the program text
2. compilation of the program
3. execution of the compiled program

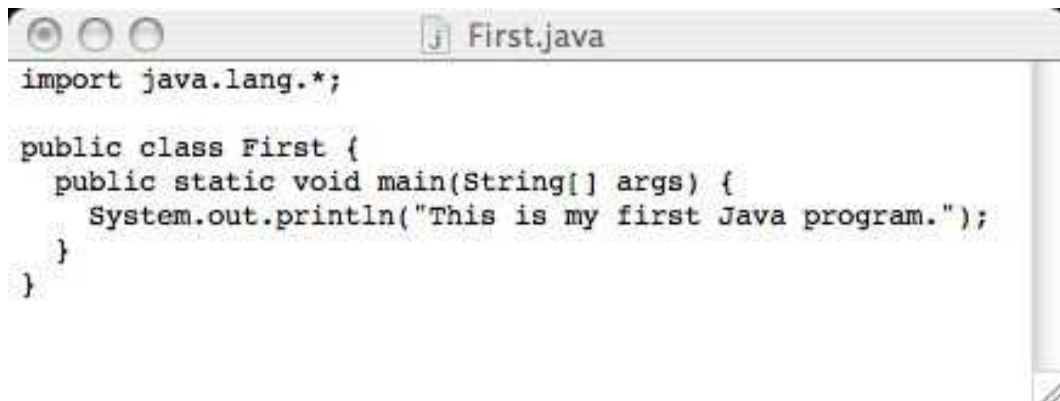
1. Preparation of the program text To prepare the program text we have to write a file containing the program. For a Java program, the name of the file has to be

ClassName.java

where *ClassName* is the name of the class defined in the program. E.g., *First.java*

The program can be written with any program that allows one to write a text file (*editor*). E.g., NotePad, Emacs, ...

Example:



```
import java.lang.*;

public class First {
    public static void main(String[] args) {
        System.out.println("This is my first Java program.");
    }
}
```

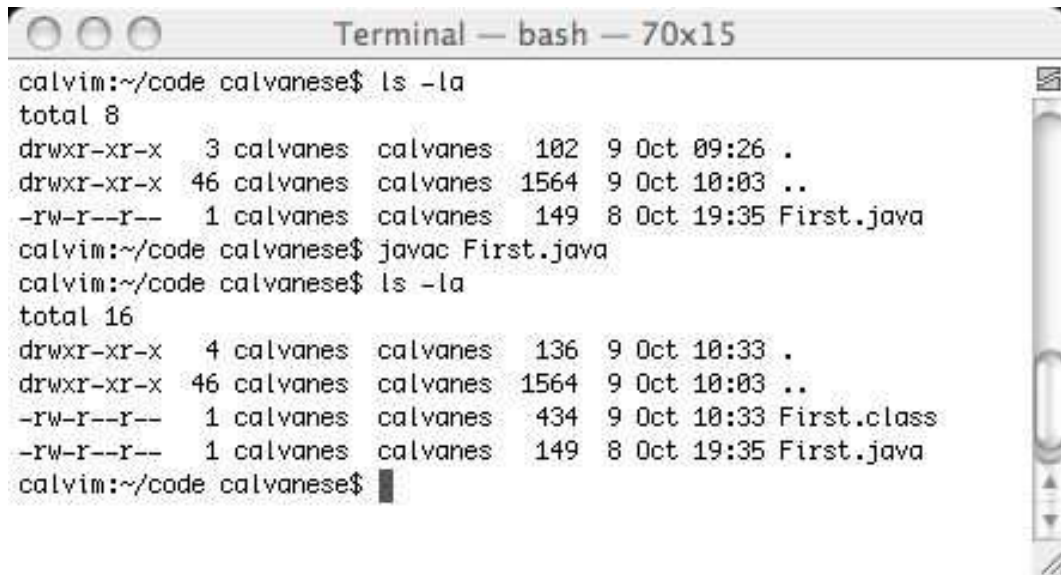
2. Compilation of the program The compilation of the program is necessary to translate the program into a sequence of commands that can be directly executed by the computer. The standard Java compiler, which is part of the Java Standard Development Kit (Java SDK), is `javac`. To use it, you have to execute the command:

```
javac ClassName.java
```

The compilation produces as a result a file called *ClassName.class*, which contains the command that can be directly executed by the computer. For example:

```
javac First.java
```

creates the file *First.class*.



```
Terminal — bash — 70x15
calvim:~/code calvanese$ ls -la
total 8
drwxr-xr-x  3 calvanese calvanese  102  9 Oct 09:26 .
drwxr-xr-x 46 calvanese calvanese 1564  9 Oct 10:03 ..
-rw-r--r--  1 calvanese calvanese  149  8 Oct 19:35 First.java
calvim:~/code calvanese$ javac First.java
calvim:~/code calvanese$ ls -la
total 16
drwxr-xr-x  4 calvanese calvanese  136  9 Oct 10:33 .
drwxr-xr-x 46 calvanese calvanese 1564  9 Oct 10:03 ..
-rw-r--r--  1 calvanese calvanese  434  9 Oct 10:33 First.class
-rw-r--r--  1 calvanese calvanese  149  8 Oct 19:35 First.java
calvim:~/code calvanese$
```

3. Execution of the compiled program A program can be executed only after it has been compiled, i.e., when we have the file `ClassName.class`.

In Java the execution of a program is done through the command

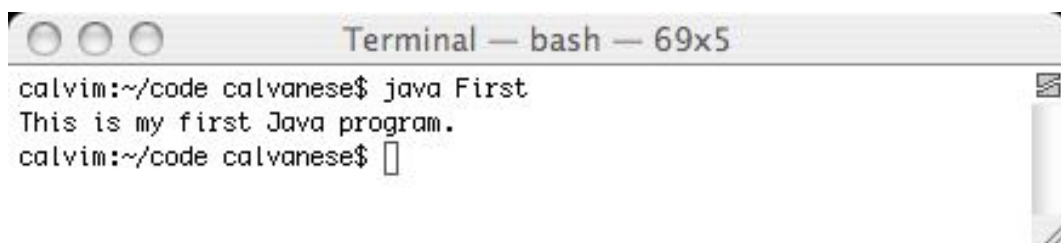
```
java ClassName
```

(without `.class`). For example, the command

```
java First
```

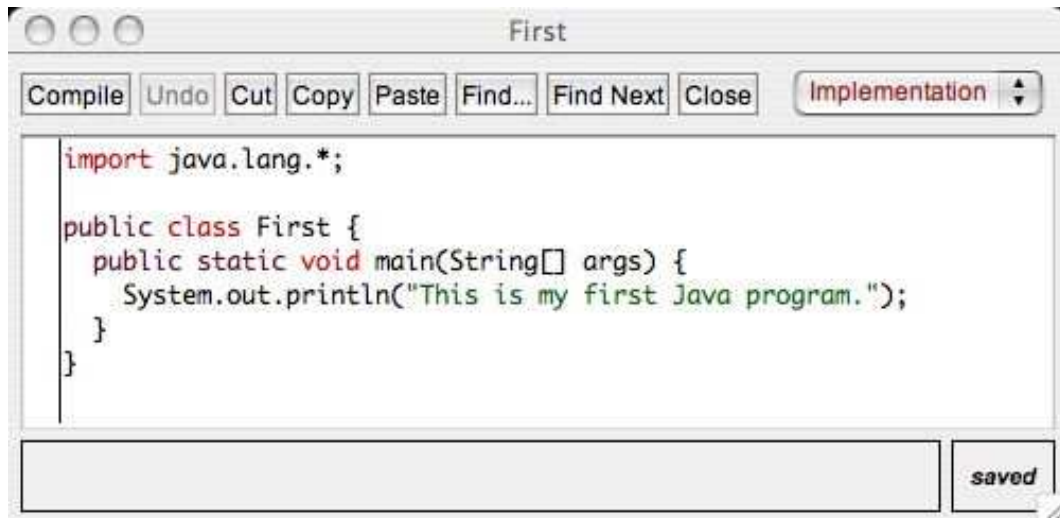
causes the execution of the program `First` (or, more precisely, of the `main` method of the class `First`), and hence prints on the screen:

This is my first Java program.

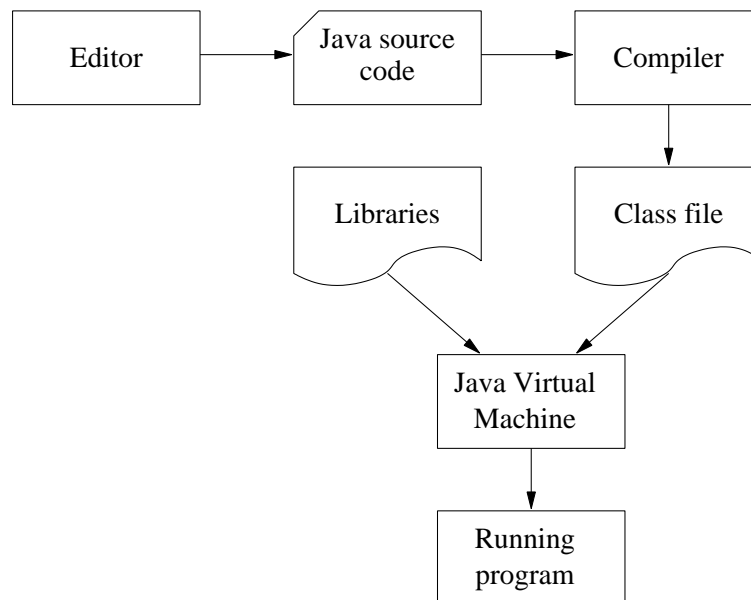


```
Terminal — bash — 69x5
calvim:~/code calvanese$ java First
This is my first Java program.
calvim:~/code calvanese$
```

Programming environment There are applications called *programming environments*, that allow one to carry out the various steps of program writing, compilation, and execution in an integrated way. Examples of Java programming environments are: JavaONE, JBuilder, JCreator, ecc. The following picture shows a screen-shot of BlueJ, a programming environment developed for teaching purposes by the Monash University, Australia, e by the University of Southern Denmark.



1.15 From the source code to the executable program (summary)



1.16 Note on the portability of Java

The Java compiler does in fact not produce code that can be directly executed by the computer. Instead it produces code that is independent of the specific computer and that is called **Java bytecode**.

Hence, the result of the compilation of a Java program is *platform independent*. This requires the use of a specific program that is able to execute the bytecode: the bytecode interpreter, known as **Java Virtual Machine**, which is activated through the command `java`.

To be able to execute a compiled Java program on some platform, it is sufficient to have the interpreter for the Java bytecode. This independence from the platform is one of the distinguishing features of Java that have led to a very quick diffusion of Java.

1.17 Errors

The following program contains various errors:

```

public class Errors {
    public static void main(String[] args) {
        System.out.println("These are my first Java errors ...")
        System.out.println("... and they will not be the last ones!!!");
    }
}

```

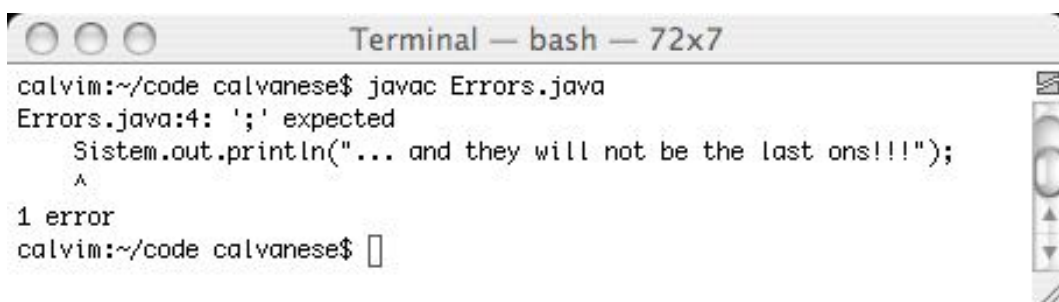
```
    }
}
```

Corrected program:

```
public class Errors {
    public static void main(String[] args) {
        System.out.println("These are my first Java errors ...");
        System.out.println("... and they will not be the last ones!!!");
    }
}
```

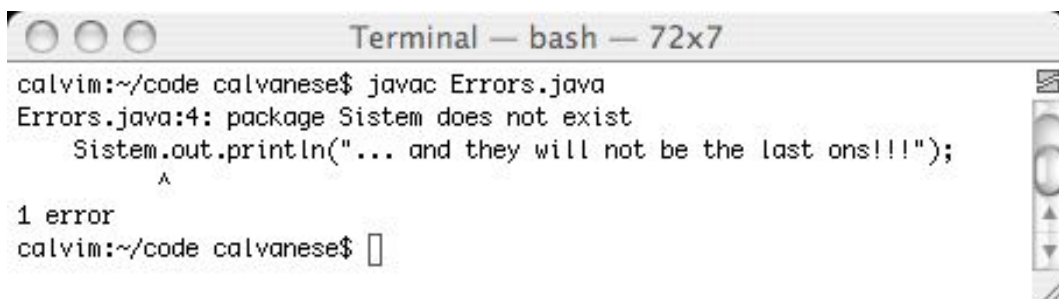
Types of errors:

- **Syntax error**, which is an error caused by a violation of the syntax rules for the language
 - example: `System.out.println(...)` – the final `';` is missing
 - syntax errors are detected by the compiler



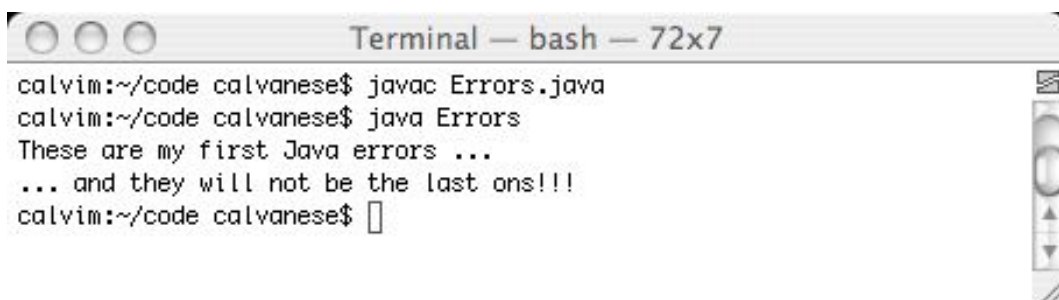
```
Terminal — bash — 72x7
calvim:~/code calvanese$ javac Errors.java
Errors.java:4: ';' expected
    System.out.println("... and they will not be the last ons!!!");
    ^
1 error
calvim:~/code calvanese$
```

- **Semantic error**, which is an error caused by the impossibility to assign a meaning to a program statement
 - example: `System.out.println(...);` – there is a spelling error in the word `System`
 - some of these errors are detected by the compiler (static semantic errors), some are detected only at runtime (dynamic semantic errors)



```
Terminal — bash — 72x7
calvim:~/code calvanese$ javac Errors.java
Errors.java:4: package System does not exist
    System.out.println("... and they will not be the last ons!!!");
    ^
1 error
calvim:~/code calvanese$
```

- **Logical error**, which is an error caused by the fact that the program realizes a different functionality than the expected one
 - example: `System.out.println("... and they will not be the last ons!!!");` – the string to be printed is not correct
 - logical errors can be detected only by analyzing the program or by testing it



```
Terminal — bash — 72x7
calvim:~/code calvanese$ javac Errors.java
calvim:~/code calvanese$ java Errors
These are my first Java errors ...
... and they will not be the last ons!!!
calvim:~/code calvanese$
```


1.18 The edit-compile-verify cycle

