02 Computing and Programming Tools & Terms

[Close all sections] Only visible sections will be included when printing this document.

▽ Computing Tools & Terms

References: L&L 1.1, 1.2

Hardware and Software

Hardware comprises all the physical, tangible parts of a computer, such as the keyboard, mouse, screen, wires, chips.

Software is everything else:
- programs and data
- a program is a series of instructions
- data is what the program instructs the computer to manipulate (numbers, words, etc.)

A computer requires both hardware and software.

The Central Processing Unit

- A CPU is also called a microprocessor
- It continuously follows the fetch-decode-execute cycle: fetch next instruction, decode it to determine action to take, execute that action, fetch next instruction, and so on.

Software Categories

- Operating System
  - controls all machine activities
  - user interface to the computer
  - manages resources such as the CPU and memory
- Application program
  - generic term for any other kind of software
  - we will write applications
- Graphical user interface (GUI)
  - some of our programs will have a GUI (although it will be very simple)

Memory

- Memory is divided into many memory locations (or cells)
- Each memory cell
  - has a numeric address, which uniquely identifies it;
  - can store a fixed number of bits (typically 8 bits, or one byte).
- Large values are stored in consecutive memory locations
- High-level programming languages, like Java, do not refer to memory locations explicitly
  - named *variables* abstract the actual memory location
  - variables have a type, so that the program knows how much memory is used and how to interpret it.

▽ Programming Tools & Terms

References: L&L 1.1, 1.4, 1.5

What is programming?

Programming is designing a *sequence of simple steps* that can be *used by a computer* to *solve some problem.*

The sequence of steps describes an *algorithm*.

What is a language? What is a programming language?

- A language has *syntax* and *semantics*. The syntax defines what words and symbols are allowed and the valid ways of arranging them (into sentences, paragraphs, etc.). The semantics define what different arrangements of the symbols *mean*.
- A programming language also has syntax and semantics. Its syntax defines valid predefined 'words' (see the list of Java reserved words) and rules for defining new words, *identifiers*, that allow the programmer to refer to new types of data (classes) or variables that hold data. Like human languages, its syntax also defines the valid ways of arranging these 'words', but it is much more strict than natural language.
- The semantics of a programming language define what the statements *do*, such as changing data in memory, or interacting with the outside world through a display device or network connection.

Syntactic and Semantic Errors

Computers are not intelligent; a program will only run if it follows *exactly* the rules of the language (syntax). A program will always do what we tell it to do, not what we *intended* it to do (semantics).
- A program that is syntactically correct is not necessarily logically (semantically) correct.
- Three types of errors:
  - Problems with syntax (compiler errors)
    - No executable program produced
  - Problems during program execution (run-time errors)
    - e.g. divide by zero, wrong sort of input
    - program terminates abnormally
  - Problem with semantics (logical errors)
    - program runs, but produces incorrect results

A simplified software development life cycle

1. Edit the source code in your text editor of choice.
2. Compile program.
   If there are **syntax errors** then goto 1.
3. Execute program and evaluate results.
   If there are **run time errors** (it 'crashes') or **semantic errors** (wrong behaviour) goto 1.

▽ Elements of Java

- Identifiers: names for variables and classes (new data types) defined by programmers (the programmers could be the authors of the standard library)
  - Rules: letters, numbers or underscore (_) only; cannot start with a number
  - Conventions: start with a lowercase letter (a–z) if a variable, uppercase letter(A-Z) if a class name; examples to come later
- Comments: provide inline documentation for a program but do not affect its behaviour
  - `//single line comment, up to end of the line`
  - `/* multiline comment, bounded by a matching closing sequence like */`
  - Java also has documentation comments (abbreviated to doc comments), which are used at the top of a program and before each method (more on them later). They look like standard multiline comments but have a different opening:

```
/** This is a doc comment. */
/**
 * And so is this, but when spanning multiple lines it's convention
 * to include a left border of asterisks.
 */
```

    You can read more about what you can do with doc comments in the Appendix on Documentation.
- Whitespace (space, tab, blank lines): Helps readability, but does not affect the program's behaviour
- Statements: program instructions terminated by a semicolon (;)
- Blocks: wrap one or more statements; define *scope* (covered later)
  - `{ }`
- Reserved words

Reserved words

These words cannot be used as identifiers (names) in Java because: they already have a defined meaning in the language (e.g., **`true`** and **`false`**); or they may be given a meaning in the future (e.g., **`const`** *might* be used for defining constant values); or they have been used in other languages in the past and cannot be used to avoid confusion (e.g., **`goto`**).

abstract	**else**	int	**strictfp**
assert	**enum**	**interface**	**super**
boolean	**extends**	long	**switch**
break	**false**	**native**	**synchronized**
byte	final	**new**	**this**
case	**finally**	**null**	**throw**
catch	float	**package**	throws
char	**for**	**private**	**transient**
class	**goto**	**protected**	**true**
const	**if**	**public**	**try**
continue	**implements**	**return**	void
default	**import**	short	volatile
do	**instanceof**	**static**	**while**
double			

A note about syntax highlighting

Program source code is just text. A .java file does not contain any formatting instructions to make some parts of it bold, or italic, or a different colour. However, source code *editors*, these notes, and the lecture slides, often present parts of the code with different formatting to indicate each element's role (e.g., a variable name, an instruction, an arithmetic operator, and so on). This is referred to as *syntax highlighting* and is done *solely* to make it easier for a human reader to comprehend the code.

A minimal Java program

This is structure of the source code for the short but valid Java program, with boxes around the parts that you can change.

```
public class ProgramNameNoSpaces {

    public static void main(String[] args) {
        Other program code...
    }

}
```

Note: This source code would need to be saved to a file named `ProgramNameNoSpaces.java` to match the name given in the code.

Tip: You can give a program a name that is any valid Java identifier, but you should follow these rules when naming programs:
- The name should start with an upper case letter (A–Z).
- If the name is made up of multiple words then they should be joined using CamelCase. So if you were writing a program to present a restaurant customer with an interactive menu you might call it `InteractiveFoodMenu` (Interctive + Food + Menu).

Tip: The program text `args` *is* actually something you could change, but the convention is to call this variable `args`.

You'll see what you can put in place of *Other program code...* (which is not valid Java by the way) over the course of the semester. In the meantime, here is the shortest valid Java program, based on the structure above. Note that it does actually *do* anything, but it will compile (if saved to `DoesNothing.java`) and run.

```
public class DoesNothing {

    public static void main(String[] args) {

    }

}
```