

# CMP-4008Y Programming I

Geoff McKeown - *Lecture Notes Week 1*

## *Introduction to Programs and Java*

### Lecture Objectives

- ◇ To describe the organization of the module.
- ◇ To introduce the high-level, object-oriented programming language Java.
- ◇ To describe the steps involved in program compilation and execution.
- ◇ To introduce `main` methods.
- ◇ To present two types of Java comments.

### Module organisation:

#### Textbook

John Lewis & William Loftus

*Java Software Solutions*

*foundations of program design*

8th edition

Pearson / Addison Wesley, 2015

ISBN-13: 978-0133594959 ISBN-10: 0133594955

#### Blackboard

- ◇ Please check it regularly!
- ◇ It will be used for announcements and distribution of material (Lecture notes, worksheets, etc.).

## Teaching Staff

- ◇ Lecturer and Module organiser:

- ▷ *Geoff McKeown*

- \* Room: S2.32

- \* email: G.Mckeown@cmp.uea.ac.uk

- ◇ *Teaching Assistants (TAs)*

- ▷ A TA is assigned to each lab group

- \* talk to your TA and try to solve problems during your lab sessions.

- \* Labs start in week 1 (i.e. today, Friday, 30th).

## Teaching Methods

- ◇ 20 hours of lectures in each semester

- ◇ 2 hours of laboratory classes per week (weeks 1 - 10)

- ◇ Self study: 3 hours per week

- ▷ reading module text

- ▷ working on problem sheets and assignments outside lab sessions

## Module Assessment

- ◇ **Formative assessment:**

During your time in the lab, you will receive verbal feedback on your work. This will help you when do your summative practical assessments.

- ◇ **Summative assessment**

This consists of THREE components: two practical assignments (worth 20% and 30% respectively, plus an end-of-year exam (50%).

- ◇ The first practical assessment will be set in week 7 and will be based on the work from the previous lab sheets.

## Low-Level and High-Level Programming

- ◇ Each computer executes the *machine code* recognised by its CPU.
- ◇ Machine-code is conceptually *very low-level*:
  - ◇ it works in *small units* both of action and of data;
  - ◇ each of these units is numerically encoded - in *binary*.
- ◇ Human beings like to work at a *higher-level* of abstraction:
  - ▷ use *names* not numbers to identify units;
  - ▷ construct larger units, of action and of data, from smaller ones, and also give meaningful names to the larger units.

## Some High-Level Programming Languages

- ◇ Over the years, many high-level programming languages have been developed, including

FORTTRAN,	Ada,	Visual Basic,
COBOL,	Modula 2,	Delphi,
Algol,	BASIC,	C++,
Prolog,	C,	C#
Pascal,	Haskell,	Java.

- ◇ Each high-level programming language consists of a specific set of words and symbols together with a set of rules for combining these words and symbols into valid *programming statements*.
- ◇ A program consists of a sequence of programming statements, which are the instructions that are carried out when the program is executed.

## Java - a High-Level Object-Oriented Programming Language

- ◇ Java is a *high-level programming language* (HLL).
- ◇ Java is an *object-oriented language* (OOL)
  - ▷ In an object-oriented (OO) language, data is used to model entities (which may be abstract or physical) from a problem domain.
  - ▷ the resulting models are called *objects*;
  - ▷ these objects are then manipulated by the program.
- ◇ The details of object-oriented programming in Java will be discussed in later lectures.
- ◇ First, you will learn how to write some simple programs to solve small computational problems.

## Executing a High-Level Language Program

- ◇ A computer can only execute its own machine code.
- ◇ So, a HLL program must (usually) be translated to machine code before it can be executed.
- ◇ This translation process is performed by a computer program called a *compiler*.
- ◇ The compiler input is called *source code*.
- ◇ The compiler output is called *object code*.

## Java Program Execution

- ◇ The Java compiler generates object code called *Java byte code*
- ◇ Java byte code is executed by a (mythical) processor called the *Java Virtual Machine* (JVM).
- ◇ A special *interpreter program*, again called the JVM, executes the Java byte code program.
- ◇ This approach allows Java programs to be executed on different kinds of CPU.

## Java Application Programs

- ◇ Every Java application program consists of one or more blocks of code called *classes*.
- ◇ All Java application programs have a class containing a **main** method:

```
public static void main (String[] args)
{
    

Some programming statements


}
```

- ◇ `public static void main (String[] args)` is the starting point of every Java application.
- ◇ Each programming statement in the `main` method is executed one after the other until the end of the method is reached, at which point the program *terminates*.

```
/*
 * A Java application to print some info about Geoff
 * Author Geoff McKeown
 */
public class PrintInfo
{
    public static void main (String[] args)
    {
        System.out.println("Geoff McKeown likes real ale");
        System.out.println("and he supports Man. Utd.");
    }
}
```

- ◇ Note that Java is a *case-sensitive language*, so `System` should always be typed with a capital S.
- ◇ `println` is a method for printing a specified string of characters to the screen:
  - ▷ where is the code for the `println` method?
  - ▷ the `println` method is invoked by the `System.out` object, which is part of the *Java Standard Library*.

- ▷ This library contains many classes whose methods make the programmer's life much simpler.
- ◇ Although `System.out` object is not part of the Java language, it is always available for use in any Java program.

## A Java Application Program with Input and Output


```
import java.util.Scanner;
/*
 * A Java application to prompt for a number and print it out
 * Author Geoff McKeown
 */
public class ReadAndPrint
{
    public static void main( String [ ] args )
    {
        Scanner scan = new Scanner(System.in);

        int number;
        System.out.println("Enter a whole number");
        number = scan.nextInt();

        System.out.println("The number you entered is:  " + number );
        System.out.println( "Finished" );
    }
}
```

- ◇ **Scanner** is a pre-defined class in the Java class library.
- ◇ A **Scanner** object allows a program to read data.
- ◇ Unlike the **System** class, the **Scanner** class is not automatically available for our use in every Java application.
- ◇ We have to ask for it to be imported if we want to use it, using an *import directive*.
- ◇ The above program creates a **Scanner** object called **scan**:

```
Scanner scan = new Scanner(System.in);
```



Standard input stream.

- ◇ **nextInt()** is a *method* of the **Scanner** class
  - ▷ allows a **Scanner** object (e.g. **scan** in the program above) to input an integer (i.e. a “whole” number).
- ◇ To input a non-integral number (e.g. 17.63) we use one of the methods **nextFloat()** or **nextDouble()**
  - ▷ non-integral numbers are represented on a computer in a *floating* point format - see later).

## Executing a Java Application

Having created an application program, how do we actually run it on a computer?

## Using the NetBeans IDE



- ◇ An integrated development environment (IDE) provides tools that support the software development process.
- ◇ Popular IDEs include:
  - ▷ NetBeans ([www.netbeans.org](http://www.netbeans.org))
  - ▷ Eclipse ([www.eclipse.org](http://www.eclipse.org))
  - ▷ JBuilder ([www.borland.com](http://www.borland.com))
  - ▷ BlueJ ([www.blueJ.org](http://www.blueJ.org))
  - ▷ jGRASP ([www.jgrasp.org](http://www.jgrasp.org))
  - ▷ JPadPro ([www.modelworks.com](http://www.modelworks.com))
- ◇ NetBeans incorporates the development tools of Sun Microsystems Software Development Kit (SDK) into a GUI-based package.
- ◇ Eclipse, promoted by IBM, is a similar IDE to NetBeans.
- ◇ Both NetBeans and Eclipse are open source projects and are available for free.
- ◇ In the NetBeans IDE, all Java development takes place within a *project*.
- ◇ A project allows a programmer to group a set of application files together.
- ◇ You will be introduced to the NetBeans IDE during your lab class this afternoon.

## Comments

- ◇ Comments are annotations inserted into the source code of a program to provide information for human readers
  - ▷ they are not processed by a Java compiler.

- ◇ Comments should be included throughout a program to provide insight into the purpose of the program.
- ◇ There are two types of comments in Java
  - ▷ *block comments*
  - ▷ *inline comments.*
- ◇ The start of a block comment is indicated by `/*`
- ◇ The end of a block comment is indicated by `*/`
- ◇ Everything between a `/*` and the corresponding `*/` is ignored by Java.

```
/*
 * This is a block comment.
 * These comments can extend
 * over multiple lines.
 */
```

- ◇ An inline comment is located on a single line.
- ◇ Its start is indicated by `//`
- ◇ Everything on a line after an `//` is ignored by Java

```
/*
 * A Java application to print some info about Geoff
 * Author Geoff McKeown
 */
public class PrintInfo
{
    public static void main (String[] args)
    {
        System.out.println("Geoff McKeown likes real ale");// first piece of info
```

```

        System.out.println("and he supports Man.  Utd."); // second piece of info
    }
}

```

## Java's Primitive Types

- ◇ There are eight primitive types in Java.
- ◇ These types are used to represent basic items of data such as whole numbers, numbers with fractional parts, characters (such as decimal digits, letters, punctuation marks), and logical values (true or false).
- ◇ The primitive types are built-in to the Java language.
- ◇ Through the use of its class mechanism, non-primitive data types may be defined
  - ▷ we have already made use of such types (e.g. the `Scanner` type).
- ◇ The four primitive types we will use most often are:
  - ▷ `int` - used to represent 32-bit integers
    - \* i.e. integers whose binary representation has no more than 32 binary digits (*bits*)
  - ▷ `boolean` - used to represent the logical values, `true` and `false`
  - ▷ `char` - used to represent characters e.g. `'a'` or `'C'` or `'3'`
  - ▷ `double` - used to represent numbers with decimal parts e.g. 17.853, in a *floating point format*
- ◇ In addition, there are three more types for representing subsets of integers:
  - ▷ `byte` - used to represent 8-bit integers
  - ▷ `short` - used to represent 16-bit integers
  - ▷ `long` - used to represent 64-bit integers

- ◇ and one more type for representing subsets of numbers with decimal parts:  
`float`
  - ▷ `double` has greater precision and a wider range than `float`.
- ◇ `int`, `double`, `char`, `boolean`, `byte`, `short`, `long` and `float` are reserved words of the Java language.
  - ▷ The words `class`, `public` and `static` and `void` used in the two Java application programs presented in the last lecture are also Java reserved words.

## Identifiers and Reserved Words

- ◇ The words that we use when writing programs are called *identifiers*.
- ◇ When writing a program we may make up words to identify items, e.g. `number` in the class `ReadAndPrint` presented in Lecture 1.
- ◇ We also use key words, such as `class`, `public`, `static`, `int` and `void`.
- ◇ Key words are reserved for special purposes in the language and can only be used in ways laid down in the rules of Java.
- ◇ We also use words that have been chosen by other programmers; for example, `System` is not a Java reserved word. But it is the name of a class in the Java standard library of pre-defined code.
- ◇ When we make up identifiers in our programs, we can use any combination of letters (upper and lower case), decimal digits, the underscore character (`_`) and the dollar symbol, but an identifier cannot start with a digit. There is no restriction (other than common sense) on the length of the identifiers we choose.
- ◇ Since Java is *case sensitive*, `number`, `Number` and `nuMber`, for example are all different identifiers.
- ◇ By convention, we start the names of classes with an upper-case letter. Other names should not start with an upper-case letter.
- ◇ Names should be chosen so as to reflect their intended usage.

## Arithmetic Operations

We can do arithmetic on numeric values using arithmetic *operators*, e.g.

$+$ ,  $-$  (addition, subtraction)

$*$  (multiplication)

$/$  (division)

$\%$  (integer remainder)

### Some Simple Arithmetic Expressions

$a + b$

$a * 9 + 6$

$(a + b) * c$

$14/5$

- (i) what is the value of this expression?
- (ii) When the division operator ( $/$ ) is applied to a pair of `ints` in Java, the result is also of type `int` and is the *integer part* of the division. So the value of  $14/5$  is 2.
- (iii) The remainder from an integer division may be obtained by using the `%` operator:
  - the expression  $14 \% 5$  gives the result 4.