COM1003 Java Programming

Dr Siobhán North
Department of Computer Science
The University of Sheffield

The Course

- This course is designed to ensure every member of the class, regardless of background, can write clear, robust, elegant, working programs in Java by the end of the year
- It starts from the assumption that you are absolute beginners

This Course is for Beginners but..

- If you know it all already
  - You still have to do the same assessments and demonstrate you can use the techniques I am assessing
  - To do well you have to be able to program to my standards, which are not necessarily the ones you are used to
  - Avoid showing off
  - But you may find some of the introductory material boring so I will let you know in advance what you can safely skip
  - Do the practical exercises anyway just in case you are not as good as you think you are

This Course is for Beginners but..

- If you can program but not in Java
  - You probably should attend all the lectures but I may be going too slowly for you at the beginning - sorry
  - Do all the practical exercises, you will have ideas to unlearn
  - Resist the temptation to patronise students who have never programmed before – of course they will learn more slowly at the beginning but they could well be better than you at the end

This Course is for Beginners

- I will be starting from scratch, assuming you know nothing
- If you have done little or no programming
  - You are not alone, whatever it may feel like at times
  - You don't need a Maths A level; you can be good at programming without being good at Maths and vice versa
  - If you have never programmed before you don't yet know how much ability you have; don't assume you lack ability just because you lack experience
  - Don't imagine you are going to learn how to programming from lectures (even mine) or books, like everyone else you must do the practical exercises

The Timetable

	Monday	Tuesday	Wednesday	Thursday	Friday
9					
10		Practical			
11					
12				Practical	
1					
2	Lecture				
3					
4	Lecture				

- The Practicals are where you will actually learn
- | will not use both the lecture slots every week

The Practical Exercises

- Every week, after the lectures, you will get a set of practical exercises to work through
- You can do them in your own time but it is better to do them in the practical slots because I will have a team of demonstrators to help and you can, and should, put your hand in the air every time you get stuck
- The demonstrators and I will be able to give you feedback on your programming style
- The solutions to the practical exercises will not be released until you have had plenty of time to do try them yourself

Assessment and feedback

- The 1st semester is worth 50% of your marks for COM1003 overall
- The 1st semester assessment is
  - Four quizzes each worth 12% of the mark
  - Three assignments worth 12%, 20% and 20% of the mark
- You will get feedback on the quizzes within 24 hours and on your assignments within three weeks
- The dates of the quizzes and assignments are on the course web page

The Course Web Page

The Course Web Page

- http://staffwww.dcs.shef.ac.uk/people/S.North/teaching/COM1003/index.html
- Google Siobhan North COM1003
- It is linked from the MOLE page but I will only use MOLE for assessment
- Once you find it bookmark it because all the supporting material for the course will appear there

Important Dates

- Quizzes all 12%
  - 15 October
  - 29 October
  - 19 November
  - 10 December
- Assignments
  - 13 Oct to 27 Oct 12%
  - 10 Nov to 1 Dec 20%
  - 1 Dec to 22 Jan 20%
- If all goes well there will be reading weeks
  - 2 to 6 November
  - 14 to 18 December
- You will need to demonstrate your final assignment code working on 21 or 22 January

How to pass this module

- Don't imagine you can learn programming from the lecture slides or a textbook – you have to do it
- Keep up with the exercise sheets
- Don't copy program code from other students because
  - you won't learn and
  - we will find out
- Attend the practical sessions
- Make good use of the demonstrators
- If you are attending all the practicals but still feel as though you are falling behind, tell me

The next few weeks for experts

- The first four weeks will not cover anything expert programmers in any language will find difficult so feel free to skip the lectures but read the lecture slides, check the web page for what I am going to cover next and do the practical sheets and assessments
- If you find you are struggling you should be at the lectures; they will be recorded and the recordings will be on the web so you can catch up retrospectively if necessary
- Make sure you attend enough of the practical sessions to be sure your programming style is OK before you hand in the first assignment
- By week five only the Java experts are excused lectures and they should start attending by week seven

The next few weeks for non-experts

- Attend everything
- Starting a 4pm this afternoon, back here when we will contemplate our very first Java program
- We will cover the “Hello world” program, variables, constants, numbers and identifiers

Beginning Java

This lecture will

- Introduce the Java programming language;
- Explain how to write a simple program that prints something out;
- Introduce the concept of programming errors;
- Introduce variables and how to name and use them;
- Present the arithmetic operators and how numerical expressions are evaluated;
- Introduce type promotion and casting;
- Introduce constants and how to name them.

Computers and programming

- A computer is a machine with, amongst other things, a microprocessor and a memory store
- A **computer program** is a set of instructions stored on the computer that it can follow in order to carry out a task
- The instructions are written in a language called a **programming language** and writing programs is what this course is all about
- Programs are written to solve problems

Algorithms

- In order to solve a programming problem, we need a step-by-step specification of the solution
- This specification is called an **algorithm**. It may be expressed in English, or in a more formal language
- The algorithm should be:
  - Unambiguous
  - Correct (finish and deliver the right result)
  - Efficient (but depends on the size of the task)
  - Robust (check for valid input data)
  - Maintainable (due to change in requirements or fixing 'bugs')

Example algorithm

This algorithm for grocery shopping is written in English-like '**pseudocode**'

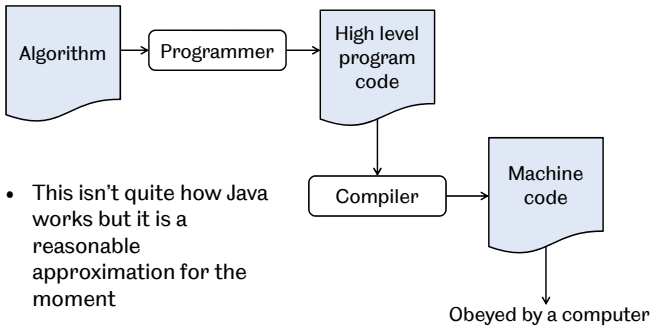
```
1. Get a trolley
2. While there are still items on the shopping list
      2.1 Get an item from the shelf
      2.2 Put the item in the trolley
      2.3 Cross the item off the shopping list
3. Pay at the checkout
```

❓ **Is this algorithm correct? Is it unambiguous? How might it fail?**

High-level programming languages

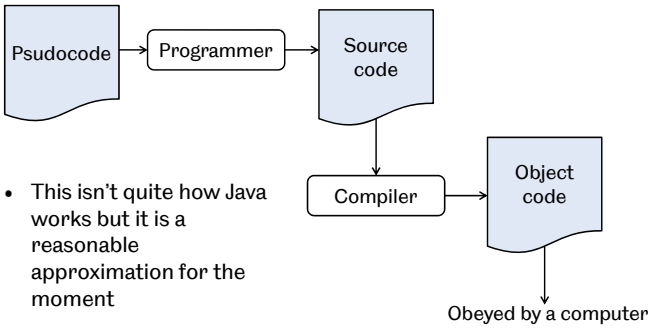
- Psudocode uses an English-like syntax that is easy for us to understand, but cannot be understood directly by a computer
- Computers understand a low-level binary language called **machine code**
- A program written in a **high-level language** must be converted to machine code before it can be **executed**
- The conversion from a high level program to machine code is normally achieved by a program called a **compiler**
- The conversion from a psudocode algorithm to a high level program is normally achieved by a **programmer**

Writing and running a program



- This isn't quite how Java works but it is a reasonable approximation for the moment

Writing and running a program



- This isn't quite how Java works but it is a reasonable approximation for the moment

A simple Java program

```
/*
A simple Java program
Written by: Guy J. Brown
*/
public class Simple {

    public static void main(String[] args) {
        System.out.print("Running a Java application");
        System.out.println("...finished.");
    }

}
```

- This program would be written in a text editor and saved with the name **`Simple.java`**

Anatomy of the Java program

- There is one **public class** in this program, called **`Simple`**
- A Java program has one publicly accessible class, and the name of this class **must** be the same as the file name of the program except that in the file name the name of the class is followed by **`.java`**
- Curly brackets **{** and **}** delimit the beginning and end of classes and methods
- The program has one **method** called **`main`**

The statements

```
public class Simple {
    public static void main(String[] args) {
        System.out.print("Running a Java application");
        System.out.println("...finished.");
    }
}
```

- Text in blue in the program above is standard to all programs
- The **statements** dictate what the program does
- Statements end with a **semicolon** and are normally obeyed in order, top to bottom
- The output is `Running a Java application...finished.`

`print` & `println`

- The following **statement**

```
System.out.println ("...finished.");
```

prints out **…finished** followed by a line break

- The statement

```
System.out.print("Running a Java application");
```

prints out `Running a Java application` without a line break

- Java doesn't print out the line until it hits a **`println`** so this statement won't work on its own

The `println` method

- The following **statement** invokes a **method** called **`println`** which belongs to an object called **`System.out`**:

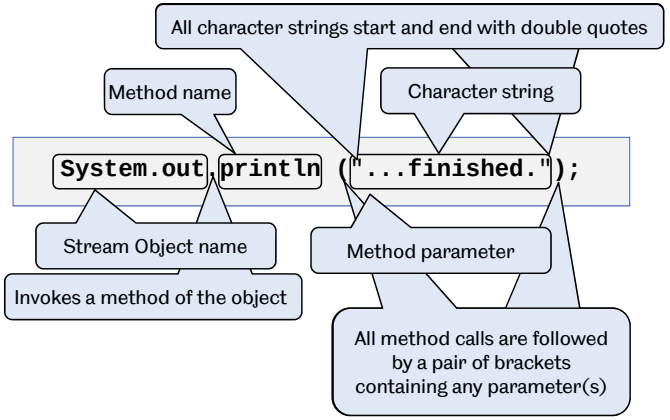
```
System.out.println ("...finished.");
```

- **`System.out`** represents a **stream** for output; anything sent to the stream appears on the screen
- The text **`"...finished."`** is an **argument** (or **parameter**) of the method. Arguments appear between parentheses. The text is a **character string**, enclosed in double-quotes

The layout of the Java program

- Lines between curly brackets which delimit the beginning and end of classes and methods should be **indented** to assist readability
- Comments, which are ignored by the compiler but are for human readers, are enclosed between the symbols **/*** and ***/** or follow the symbol **//**
- Comments should always be included
- Blank lines and spaces (except in the middle of words or numbers) are also ignored but should be used to improve readability

The println method



The println method

- The following **statement** invokes a **method** called **println** which belongs to an object called **System.out:**

```
System.out.println ("...finished.");
```

- **System.out** represents a **stream** for output; anything sent to the stream appears on the screen
- The text **"...finished."** is an **argument** (or **parameter**) of the method. Arguments appear between parentheses. The text is a **character string**, enclosed in double-quotes

print & println

- The following **statement**

```
System.out.println ("...finished.");
```

prints out **…finished** followed by a line break

- The statement

```
System.out.print("Running a Java application");
```

prints out **Running a Java application** without a line break

- Java doesn't print out the line until it hits a **println** so this statement won't work on its own

The statements

```
public class Simple {
    public static void main(String[] args) {
        System.out.print("Running a Java application");
        System.out.println("...finished.");
    }
}
```

- Text in blue in the program above is standard to all programs
- The **statements** dictate what the program does
- Statements end with a **semicolon** and are normally obeyed in order, top to bottom
- The output is `Running a Java application...finished.`

Anatomy of the Java program

- There is one **public class** in this program, called **`Simple`**
- A Java program has one publicly accessible class, and the name of this class **must** be the same as the file name of the program except that in the file name the name of the class is followed by **`.java`**
- Curly brackets **{** and **}** delimit the beginning and end of classes and methods
- The program has one **method** called **`main`**

A simple Java program

```
/*
A simple Java program
Written by: Guy J. Brown
*/
public class Simple {

    public static void main(String[] args) {
        System.out.print("Running a Java application");
        System.out.println("...finished.");
    }

}
```

A **comment**

- Comments are for human readers only they are ignored by the compiler

Dealing with errors

- Sometimes compiler messages are not so clear. If we remove the first quote from line 9 of the **`Simple.java`** program, the following error report is generated:

```
U:…>javac Simple.java
Simple.java:9: ')' expected
       System.out.print(Running a Java application");
                                ^
Simple.java:9: unclosed string literal
       System.out.print(Running a Java application");
                                                    ^
2 errors
```

Exercise

What is displayed by the following program?

```
/*
   A fun exercise
   Written by: Guy J. Brown
   First written: 19/8/02
*/
public class FunExercise {
   public static void main(String[] args) {
       System.out.print("Java programming");
       System.out.println(" is");
       System.out.print("F");
       System.out.print("U");
       System.out.println("N");
   }
}
```

```
U:…>java FunExercise
Java programming is
FUN
```

Variables

- Computer programs often store and manipulate numbers, words and symbols
- We use **variables** to act as storage boxes for information. We can set, retrieve and modify the value of a variable
- Every variable has an **identifier**
- Every variable has a **type**, which indicates the kind of information it holds and how much computer memory is required to store it
- Variables have to be **declared** – this creates space to store its value, and associates an identifier and a type with the space

Identifiers

- The name of a variable, or anything else in Java, should be chosen for clarity, e.g. **`numberOfBooks`** rather than **`x`**
- Identifiers should always start with a letter but after the first letter they can contain any sequence of uppercase or lowercase letters, digits and the underscore character '_'
- Java is **case sensitive**, so **`numberofbooks`** and **`numberOfBooks`** are different identifiers in Java
- By convention variable names begin with an initial lowercase letter (e.g. **`width`**)
- By convention variable names are in **camel case**; all words except the first start with a capital and there are no underscore symbols

Reserved words

These **reserved words** cannot be used as identifiers:

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

Exercise

Which of the following are valid identifiers? Which are conventional variable names?

- jamesbond007
- DOUBLE
- x2
- high_score
- Identifier
- 2beOrNot2Be
- numberOfWindows
- AC/DC
- homer simpson
- low-score
- numberofwindows
- _identifier

Basic Types

- Every variable has a type, the type of the value it holds
- Some of the simplest types are numbers
- There are two kinds of numbers:
  - integers (whole numbers such as 42)
  - real numbers (contain a decimal point, such as 3.141592)
- In Java the most commonly used numeric types are `int` for integers and `double` for real numbers
- These are two of the **basic types** in Java

Declarations and assignment

- This declares the variable **`heightInInches`**; it creates space to store an integer and associates the identifier **`heightInInches`** with the storage space

```
int heightInInches;
```

- This sets the value of **`heightInInches`** to 72

```
heightInInches = 72;
```

- This will print out 72

```
System.out.println(heightInInches);
```

Declaration and assignment

- A variable is only **declared** once, although its value can be changed many times
- Values are placed in a variable by **assignment**
- The assignment operator is '=', which should be read as 'takes the value of'
- We can declare several variables of the same type in the same statement, with their names separated by commas

```
double width, length = 4.0, area;
width = 6.0;
length = 3.0;
```

Example - the area of a field

By convention class names start with an upper case letter

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

        double width = 3.2;   // width of field in metres
        double length = 7.8;  // length of field in metres

        // compute the area
        double area = width*length;

        // write the result
        System.out.print("Your field has an area of ");
        System.out.print(area);
        System.out.println(" metres squared.");
   }
}
```

Multiplication sign

```
Your field has an area of 24.96 metres squared
```

More on assignment and declaration

- We refer to a variable by writing its name:

```
System.out.println(area);
```

- Likewise, we can copy the value of one variable to another:

```
length = width;
```

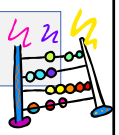
Expressions and arithmetic operators

- We form expressions using **arithmetic operators**:

```
int y = 7;
int x = y+2;
int a = y-2;
int b = y*2;
int c = y/2; // integer division, c is 3
int d = y%2; // modulus (remainder), d is 1
```

- The '/' operator gives different behaviour for **double**:

```
double y = 7.0;
double c = y/2.0; // c is 3.5
```



Short form notation

- A variable can appear on both sides of an assignment because Java works out the value on the right of the assignment before assigning it to the variable on the left
- So this

```
sum = sum+2;
```

means **sum** becomes 2 more than it was before

- There is a short form meaning the same thing

```
sum += 2;
```

Short form notation

- To add one to a variable, we can write

```
count++;
```

- Similarly, to subtract one from a variable:

```
count--;
```

Summary of short form notation

Long form	Short form
`x = x + 3;`	`x += 3;`
`x = x - 7;`	`x -= 7;`
`x = x * 2;`	`x *= 2;`
`x = x / 4;`	`x /= 4;`
`x = x % 6;`	`x %= 6;`
`x = x + 1;`	`x++;`
`x = x - 1;`	`x--;`

Example - wall paper and carpet

```
public class WallPaper {
   public static void main(String[] args) {
        int length=2, width=4, height=5;
        int carpetSize, wallpaperSize;
        // do the calculations
        carpetSize = length*width;
        wallpaperSize = 2*height*(length+width);
        // print the result
        System.out.print("Your room needs ");
        System.out.print(carpetSize);
        System.out.println(" square metres of carpet and");
        System.out.print(wallpaperSize);
        System.out.println(" square metres of wallpaper");
   }
}
```

```
Your room needs 8 square metres of carpet and
60 square metres of wallpaper
```

Precedence rules

- Java decides the order in which operations are carried out in an expression according to **precedence rules**
- Multiplication and division have a higher precedence than addition and subtraction
- In an expression with no brackets, operations with a higher precedence are performed first
- If the expression contains brackets, each pair of brackets is evaluated in turn, starting with the innermost pair
- For operators with equal precedence, Java works from left to right
- When in doubt, use extra brackets!

Precedence levels

Precedence level	Operators
1	unary +, unary -, ++, --
2	*, /, %
3	+, -
...	
10	=, +=, -=, *=, /=, %=

Unary operators appear in expressions such as +4 and -x.
Hence, -4 - -5 means (-4) - (-5)

Numeric Types

- The type of a variable is important in arithmetic expressions

```
int i=14;              // i takes the value 14
i=i/5;                 // i takes the value 2
double d=14;           // d takes the value 14.0
d=d/5;                 // d takes the value 2.8
```

- So is the type of a **literal value** – a number which appears directly in the program

```
int i=14/5;            // i takes the value 2
double d=14/5;         // d takes the value 2.0
double e=14/5.0;       // e takes the value 2.8
```

Mixing types

- Care is needed when using expressions with mixed types:

```
int first=12, second=9;
double average = (first+second)/2;
```

returns 10.0 in **average** because of the integer division. To fix this, we can force real division:

```
double average = (first+second)/2.0;
```

Or use an explicit **cast** to temporarily change the type:

```
double average = (first+(double)second)/2;
```

This casts the variable **second** into a double

Truncation and Rounding

- When a real number is assigned to an integer it is **truncated** – its decimal part is removed
- If you want to round it to the nearest integer use **`Math.round(..)`**

```
int a;
double d = 2.9;
a = (int)d; // a is truncated to 2
a = (int)Math.round(d); // a is rounded to 3
```

- We will come back to **`Math`** later

Constants

- Values that don't change during program execution should be declared as **constant**:

```
final double WIDTH = 3.2; // width in metres
```

- The compiler will complain if you try to assign a new value to a **final** variable (constant):

```
error Can't assign a value to a final variable
```

- By convention, use upper case words separate by an underscore for constant names:

  `VAT_RATE` `WIDTH_OF_ROOM`

Constants aid software maintenance

- Constants make a program easier to maintain. This is bad:

```
double taxPayable = price*20/100.0;
```

- This is easier to read and maintain:

```
final double VAT_RATE = 20; // percentage tax
...
double taxPayable = price*VAT_RATE/100.0;
```

- A constant value may be used many times in a program; if you define it, you only need to change it once.

Example – using constants

```
public class FeetInches {
    public static void main(String[] args) {
        final double CM_PER_INCH = 2.54;
        final int INCHES_PER_FOOT = 12;
        int numFeet = 3;
        int numInches = 2;
        double numCm = CM_PER_INCH *
                (numInches + numFeet*INCHES_PER_FOOT);
        System.out.print(numFeet);
        System.out.print(" feet and ");
        System.out.print(numInches);
        System.out.print(" inches is ");
        System.out.print(numCm);
        System.out.println(" cm");
    }
}
```

```
3 feet and 2 inches is 96.52 cm
```

Summary of key points

- The simplest Java program consists of a class containing a main method which contains statements to be obeyed in order and is stored in a file with a name that matches that of the class
- **`System.out.print`** and **`System.out.println`** print things out
- Comments, spaces and indentation are important
- Variables can be thought of as named boxes in the computer's memory that store data of a specified type
- Constants are like variables but the value cannot be changed
- Java can do arithmetic but be careful of precedence and the difference between integer and real division