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 all absolute beginners

The Timetable

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

See later

Only today

But sometimes we will use this slot for Quizzes

Every week but the venue will change

Assessment

- The assessment is the same for everyone, expert or beginner
- 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 first semester mark

Feedback

- You will get feedback on your programming style at the practical sessions
- You will get feedback on the quizzes within 24 hours
- You will get feedback on your assignments within three weeks

Important Dates

- Quizzes all 12%
  - 16 October
  - 6 November
  - 20 November
  - 4 December
- Assignments
  - 10 Oct to 27 Oct 12%
  - 7 Nov to 1 Dec 20%
  - 14 Nov to 19 Jan 20%
- If all goes well there will be reading weeks
  - 30 October to 3 November and
  - 11 to 15 December
- Assignment 3 demonstration 17-19th January

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 mostly use MOLE for assessment
- Once you find it, bookmark it because all the supporting material for the course will appear there

The Course Web Page

The Practical Exercises

- After the lecture every week you will get a new set of practical exercises
- You will learn to program by doing them
- You will eventually get solutions but only when you have had plenty of time to do the exercises without seeing the solution
- Solutions are never as helpful as you imagine they are going to be

The Practical Sessions

- This is when you can work through the practical exercises with someone around to help when you get stuck
- This is when you will get feedback on your programming style
- If you can't finish the practical sheet in the practicals you will need to work at home too
- Alternatively you can do all the practical work at home but get your style checked

The Timetable for Beginners

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

Attend both

- There is only one exercise sheet per week whichever practical you go to

The Timetable for Experts

	Monday	Tuesday	Wednesday	Thursday	Friday
9					
10					
11		Practical Computer Room 3			
12				Style Surgery	
1				Room 3	
2					
3					
4					

Skip both provided you can do the exercises on your own

This is to allow expert programmers to get feedback on their style only, if your program doesn't work you should be in the practicals

Jessop West, Seminar room 1

The Timetable for Everyone Else

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

Attend the one on your timetable

- There is only one exercise sheet per week whichever practical you go to

If you know it all already…

- The important thing to remember is that the marking scheme is the same for everyone
- You have to demonstrate you can use the techniques I want to be sure the beginners have mastered
- So if I say students have to use a particular technique and there are marks associated with using that technique you have to use it to get the marks even if you know better ways of doing things

If you know it all already…

- Do the practical exercises anyway just in case you are not as good as you think you are
- Go to one of the style surgeries and get your style checked
- Read the marking scheme for every assignment, a program that works will not be good enough

If you can program but not in Java

- You probably should attend all but the first few 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
- Unless you take to Java like a duck to water go to one or other of the practical sessions
- If all your programs work go to the style surgery to get your style checked

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

If you have done little or no programming

- Don't imagine you are going to learn how to program from lectures (even mine) or books, like everyone else you must do the practical exercises

- I couldn't teach you to ride a bicycle by lecturing to you and I can't teach you to program, you have to learn by practice

How to pass this module

- 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 enough of the practical sessions
- Make good use of the demonstrators when you are stuck

Who can skip lectures

- Beginners should attend everything starting with the rest of this lecture
- Experts in any language can probably manage without the lectures for the first 3 or 4 weeks
- Experts in Java can probably skip up to week 7
- Everyone should do all the practical sheets

If you overestimated your ability

- The lectures will be recorded and the recordings will be on the web as will the lecture slides so you can catch up retrospectively if necessary
- Make sure you do the practical sheets in sync with reviewing the lectures
- Come to the practical sessions to catch up; you don't have to restrict your questions to the sheet for the current week

Beginning Java

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

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

An 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?

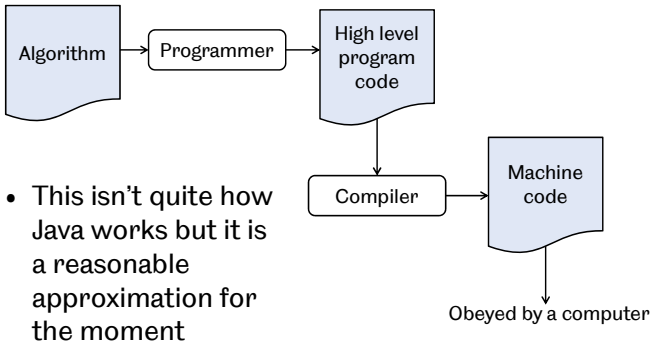
Algorithms and computers

- An algorithm written in psudocode has to get into the computer some how
- 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**
- In between we have a **high level programming language**

High-level programming languages

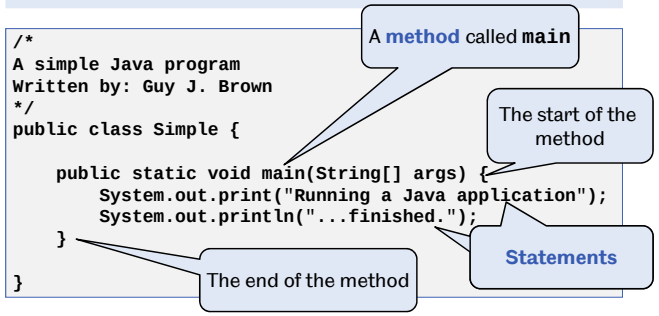
- A program written in a **high-level language** must be converted to machine code before it can be **executed**
- The conversion from a high level program to machine code is normally achieved by a program called a **compiler**
- The conversion from a psudocode algorithm to a high level program is normally 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

A simple Java program



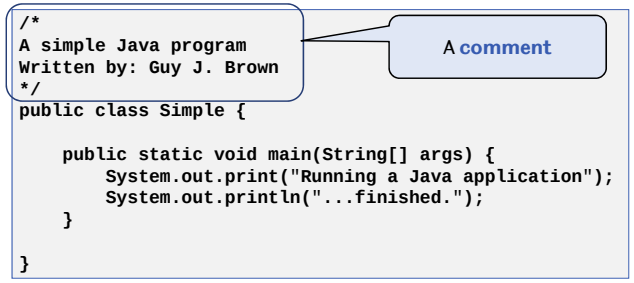

```
/*
A simple Java program
Written by: Guy J. Brown
*/
public class Simple {

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

}
```

- The simplest Java program consists of a single public class with a single `method` which is always 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.");
    }

}
```

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

Anatomy of the Java program

- The java program is stored in a file whose name ends `.java`
- It has one publicly accessible `class` whose name is whatever precedes `.java` in the file name
- It has one `method` which is always called `main`
- Curly brackets `{` and `}` delimit the beginning and end of classes and methods

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

print and println

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

- **print** prints out whatever is in the brackets following the word **print**
- **println** prints out whatever is in the brackets following the word **println** and then prints a line break, it moves the cursor to then next line
- Mostly you will need **println**

The println method

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

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

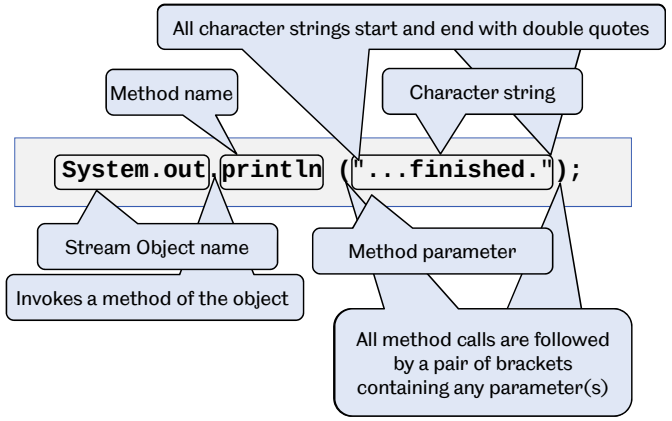
- **System.out** represents a **stream** for output; anything sent to this stream appears on the screen

The println method

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

- The text **"...finished."** is an **argument** (or **parameter**) of the method **println**
- Arguments appear between brackets **(** and **)**
- The text is a **character string**, enclosed in double-quotes

The println method



Layout matters

- These programs do exactly the same thing

```
/*
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.");
    }

}
```

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

Compiling and running java

- Write your program in a text editor and save it
- Compile it using the **Java Development Kit** (JDK)

```
U:…>javac Simple.java
```

- If nothing goes wrong, a file called **`Simple.class`** will be created
- To run it type

```
U:…>java Simple
```

Errors

- Errors in a program may occur:
  - At **compile time** (syntax errors and warnings)
  - At **run time** (e.g., out of memory error)
- Even if it executes, a program may not do what was intended due to a **logic error** in the design

Dealing with errors

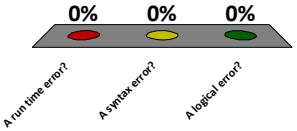
- If we remove the semicolon at the end of line 10 of our program, the compiler stops with the following error message:

```
U:…>javac Simple.java
Simple.java:10: ';' expected
      System.out.println("…finished.")
                                      ^
1 error
```

Is this error…

```
U:…>javac Simple.java
Simple.java:10: error';' expected
      System.out.println("…finished.")
                                      ^
1 error
```

A. A run time error?
✓B. A syntax error?
C. A logical error?



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
```

Variables

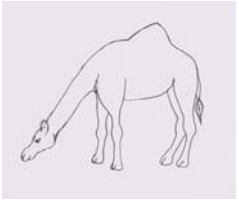
- Computer programs store and manipulate information such as numbers and words
- **Variables** act as named storage boxes for information of a particular type
- Values can be put into the box (variable) and retrieved when necessary
- Variables have to be **declared** – this creates space to store its value, and associates an **identifier** and a **type** with that space

Identifiers

- Identifier should always be meaningful e.g. **`numberOfBooks`** rather than **`x`**
- Java is case sensitive, so **`numberofbooks`** and **`numberOfBooks`** are different identifiers in Java
- Identifiers must start with a letter but after that can contain any sequence of uppercase or lowercase letters, digits and the underscore character '_' but no spaces

Variable Identifiers

- All identifiers start with a letter but by convention variable names begin with a lowercase letter (e.g. **`width`**)
- By convention variable identifiers are in **camel case**; mostly lower case but all words except the first start with a capital and there are no underscore symbols (e.g. **`numberOfBooks`**)



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
- For instance numbers can be
  - integers (whole numbers such as 42)
  - real numbers (contain a decimal point, such as 3.141592)
- Two of the **basic types** in Java are **`int`** for integers and **`double`** for real numbers

Variable declarations

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

```
int heightInInches;
```

- More than one variable of the same type can be declared at once with the identifiers separated by commas

```
int heightInInches, heightInCms;
```

Assignment

- Once the variable has been declared this sets its value to 72

```
heightInInches = 72;
```

- And then this will print out 72

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

because we refer to a variable's value by its identifier

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 assign values to variables as we declare them

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

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.
```

Arithmetic

- This statement

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

  is a declaration and an assignment but the value assigned is calculated

- The assignment operator can have complicated arithmetic expressions on its right but it always has a single variable name on the left

Expressions and arithmetic operators

- We form expressions using arithmetic operators:

```
int y = 10;
int a = y+2; //afterwards a is 12
int b = y-3; //afterwards b is 7
int c = y*4; //afterwards c is 40
int d = y/5; //afterwards d is 2
int e = y%6; //afterwards e is 4
```

- The % operator works out the modulus (remainder)

Integer division

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

```
int count = 17;
double size = 17.0;
System.out.println(count / 2); //prints out 8
System.out.println(size / 2);  //prints out 8.5
```

- Integer division truncates the result

```
System.out.println(count / 16); //prints out 1
```

- Integer division only happens if there is an integer on either side of the /

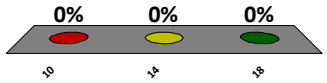
Precedence rules

- In complicated expressions Java decides the order in which operations are carried out according to **precedence rules**
- It does multiplication, division and modulus before addition and subtraction
- Where operators have the same precedence (e.g. *, / , %) it works left to right
- If there are brackets it works out the contents of the brackets before everything else
- When in doubt, use extra brackets!

Given the declarations `int a=3, b=4, c=5;`

What is the value of the expression `2*a+b;`

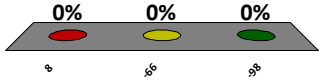
✓A. 10
B. 14
C. 18



Given the declarations `int a=3, b=4, c=5;`

What is the value of the expression `2*((a+b)*-c+2);`

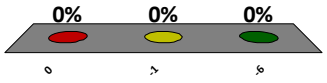
A. 8
✓B. -66
C. -98



Given the declarations `int a=3, b=4, c=5;`

What is the value of the expression `-a*b/c*a;`

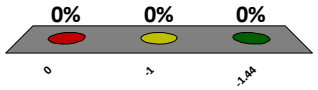
A. 0
B. -1
✓C. -6



Given the declarations `double x=2.0, y=1.2;`

What is the value of the expression `-x*y/x*y;`

A. 0
B. -1
✓C. -1.44



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;
```

puts 10.0 in average because of integer division.

- To fix this, we can force real division:

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