Java programs

- Made up of classes
- Each class in a file
- The name of the class must match the name of the file
- The name of the class must start with a capital letter
- Text files with .java extension

Compilation

- Program written in high level language (text files)
- Is compiled using IDE or command line compiler
- The compiled files are .class, one for each java file

STRUCTURE OF A JAVA PROGRAM:

```
public class _________ {

   Optional variable declarations and intialization

   Optional methods

}
```

The name of the class must be exactly the same as the name of the file in which the class is written

STRUCTURED PROGRAMMING

Methods are also called subroutines:

Named blocks of code that perform some task.
Evolution from having line numbers and using **`go to`** statements

Unstructured programming

```
1  read a
2  if a>3 go to 10
3  z=a-1
4  y=a+8
5  print y
6  if y<25 go to 13
7  print "hello"
8  end
9  print x
10 x=a+2
11 if x-1>10 go to 4
12 go to 6
13 z=a+1
14 print z
```

Spaghetti code

Structured programming

```
input a;
input b;
c=do_task1(a,b);
if c>1 then do_task2();
```

task1 and task2 are subroutines that perform some task.

Go, execute, come back.

COMPLETE FIRST PROGRAM

```
public class HelloWorld {

// at this point we have no variable declarations

// we have one method:

   public static void main(String[] args){
      System.out.println(“Hello World”);
   }

}
```

- If a class has a **main** method then that method is used by the Operating System as first point of access into the program.
- // are used to write single line comments and are ignored by the compiler.
- /* */ are used to write longer comments.
- The **System.out.println** instruction is used to print something to the console.

COMPILATION DETAILS (IDE)

The program called HelloWorld must be written in a file called:

```
HelloWorld.java
```

- When using an IDE the details of the name of the file might be hidden from the user
- In Eclipse, when the green arrow is selected (run) the program is compiled automatically and the HelloWorld.class file is generated. Again these details are hidden from the end user
- In Eclipse, the HelloWorld.class file is executed after the green arrow has been pressed.

COMPILATION DETAILS (Command line)

Compile using the *javac* command:

```
C:> javac HelloWorld.java
```

Once it has been compiled the byte code HelloWorld.class can be executed using the *java* command:

```
C:> java HelloWorld
HelloWorld

C:>
```

IDENTIFIERS

- Used for naming: variables (objects), classes
- Names of classes always capitalized
- Other objects with lower case
- Cannot start with a number (0, 1, 2, …)
- Underscore (_) is fine
- Names can be long.
- For readability the name should be descriptive of what the identifier is for.
- Underscore can be used to separate words like:

```
maximum_grade
```

- Although using capital letters to separate words is more widely used:

```
maximumGrade
```

- Each company might define its own conventions (for human readability and compatibility)

METHOD DECLARATION (HOW TO WRITE ONE)

We already wrote one method:

```
public static void main(String[] args){
   System.out.println(“Hello World”);
}
```

A method as a **Header** (first line) that includes:

- Properties (public, static) that we will talk about later.
- A type (void) that tells us what type of result is expected from it.
- A name that follows the conventions of an identifier that we described earlier
- Parameter list. More about this later.

The method also has a **Body**:

- The text that goes between the { and the } is called the body of the method
- The body includes those instructions that must be executed.

METHODS (HOW TO USE ONE)

When we use a method we say that the method is called.
Calling a method involves three steps:

- Control is transferred to the beginning of the method
- The method executes
- Control is returned to the next instruction after the method was called.

We used a method in our example:

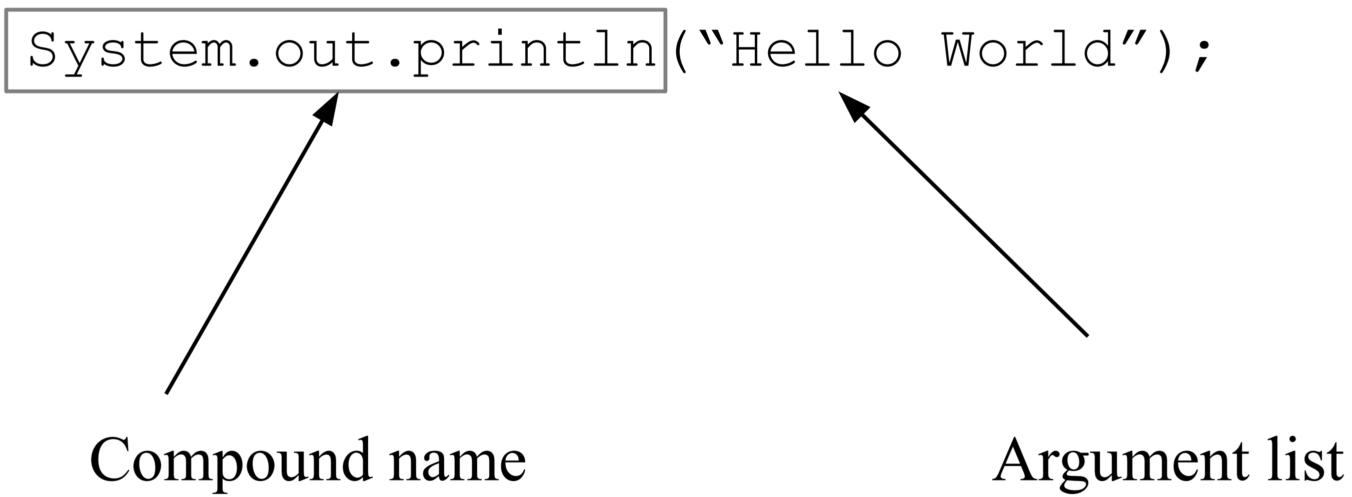
```
System.out.println(“Hello World”);
```

This method is used to display something to the console:

Displays “Hello World”, and a new line
Returns to where it was called

PARTS OF A METHOD CALL

Not to be confused with the parts of a Method declaration!



```
System.out.println(“Hello World”);
```

Compound name

Argument list

EVERY STATEMENT IN JAVA MUST END WITH
SEMICOLON ;

VARIABLES

Are used in Java to represent the contents of a memory location.
Variables have:

- Name (follow identifier conventions)
- Type (one of the primitive types or classes… later)
- Value (contents)

Remember the 8 primitive types: byte, short, int, long, float, double, boolean, char

LITERALS

Constant piece of data.
Literals have:

- Type
- Value (contents)

VARIABLE DECLARATION AND LITERAL EXAMPLES

```
int x=3;
```

NAME:	x
TYPE:	int
VALUE:	3

Somewhere in memory

```
double y=3.141516;
```

NAME:	y
TYPE:	int
VALUE:	3.141516

Somewhere in memory

```
4
```

TYPE:	int
VALUE:	4

In the case of literals, the type is implied by the value and is not always what we have in mind...

VARIABLE DECLARATION

Every variable MUST be declared before it is used.

```
int n,m,k;
float y;
double myVariable;
char ch1;
```

Several variables at the same time

They can also be initialized at the same time:

```
int n = 4;
float y = 254.8;
double myVariable = 3.1415926535897932;
char ch1 = ‘A’;
```

DEFAULT VALUES FOR EACH DATA TYPE

If variables are not initialized when declared, a default value is assigned to them according to their type:

Data type	Default Value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	u0000
boolean	false

OPERATORS

Operators act on:

- Variables
- Literals

Operators produce literals

Binary arithmetic operators:
+, -, /, *, %

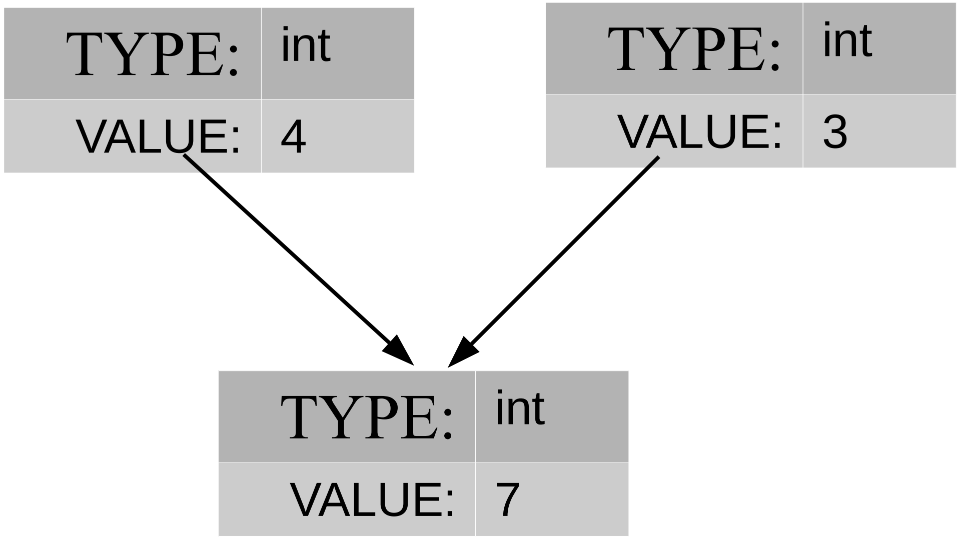
The operators can only act on two (literals or variables) at one time (the processor cannot add three numbers at once, it must proceed two at a time).

LIST OF BINARY ARITHMETIC OPERATORS

Operator	Description	Example
+ (addition)	Add left hand side to right hand side	3 + 4 gives 7
- (subtraction)	Subtracts right hand side form left hand side	50 - 23 gives 27
* (multiplication)	Right hand side times left hand side	3 * 8 gives 24
/ (division)	Divides left hand side by right hand side	200 / 2 gives 100
% (modulus)	Remainder of dividing left hand side by right hand side	17 % 3 gives 2

BINARY ARITHMETIC OPERATORS

Literal or variable + Literal or variable



TYPE:	int
VALUE:	4

TYPE:	int
VALUE:	3

TYPE:	int
VALUE:	7

Same idea with the other basic operators: -, *, /, %

How is the type computed?