Fundamentals of Programming 1

Lecture 1 – Introduction to Java Programming Language

Aurelia Power, TU Dublin – Blanchardstown Campus, 2019

* Notes based on the Java Oracle Tutorials, Deitel & Deitel, and Horstman

Assessment

A. **Continuous Assessment (CA)** carried out throughout the semester – **50%:**
   1. **MCQ 1– 15%**
   2. **MCQ 2 – 15%**
   3. **Practical exam – 20%**

B. **Formal Written Exam** sometimes during the January examination period – **50%**

- Dates to be confirmed as soon as possible.

Weekly Hours and Some Ground Rules

- **2 hours lecture** – all students attend in the same time on Monday from 9.00 to 11.00.
- **2 hours practical work** in the lab – each individual group has its own designated time.
- **NOTE:** before each lab session, you must go over the lecture notes, as each lab is based on a corresponding lecture and it aims to allow you to practice and implement concepts introduced during the lecture.
- All material introduced in the module is examinable, whether in the lecture or lab.

Weekly Hours and Some Ground Rules

- Mobiles phones and personal devices must be switched off; you are welcome to bring your own laptops during labs (not during the lecture).
- You must not shout, use profane language or engage in any disruptive/disrespectful/inappropriate behaviour during both lecture and labs.
- Ask questions relevant to the module and try and engage as much as possible.
- Use the discussion forum to post any programming problems.
- Enrolment Key:

  2019_FOP2

The Java Technology

- **James Gosling** and co. officially introduced Java in 1995;
- Java is used for creating web pages, develop desktop applications, develop large-scale enterprise applications, enhance the functionality of web-servers, develop consumer-devices applications, etc.
- The **Java technology** is both:

- A high-level programming language
- A platform.

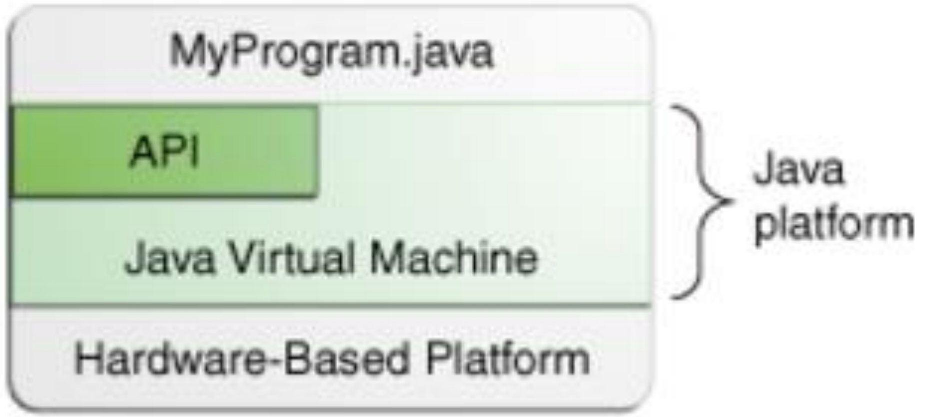
- The **Java Platform** has **2 components**:

1. The Java Virtual Machine (JVM)
2. The Java Application Programming Interface (API)

The Java Platform

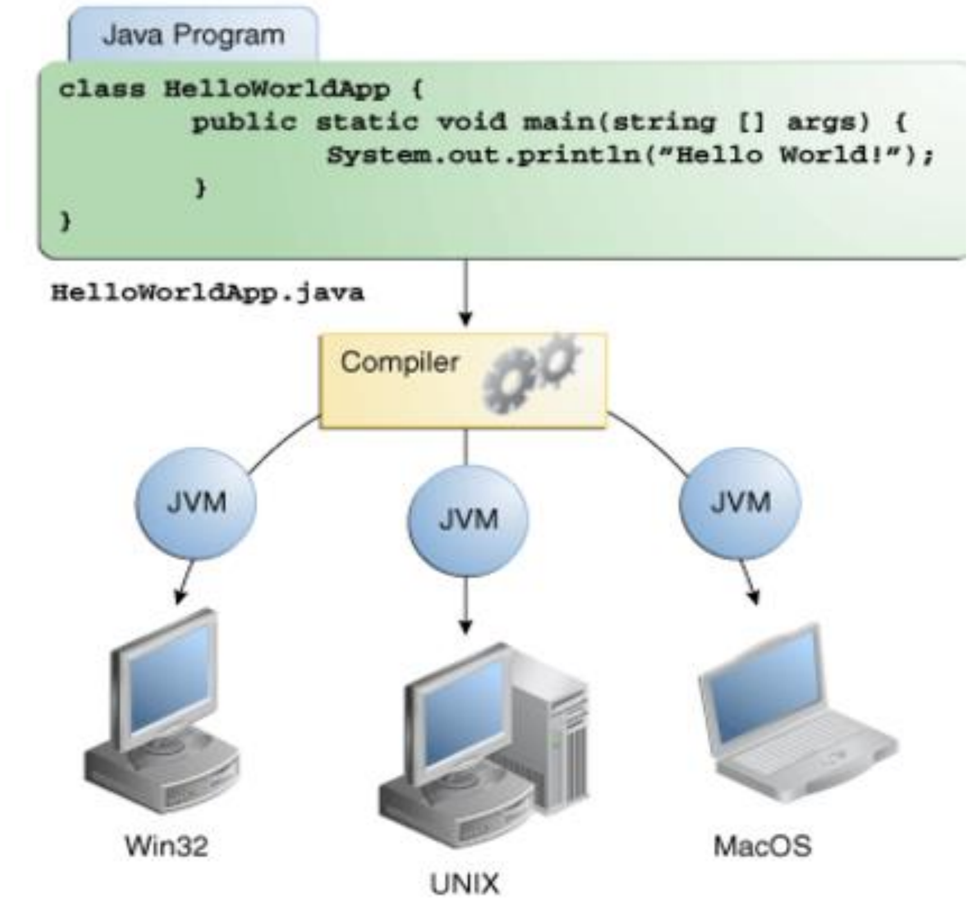
1. The **JVM** provides the base that is ported on various hardware-based platforms; it is a software application/ an abstract computing machine that hides the implementation of the program from the underlying platform/operating system.
2. The **Java API** is a collection of ready-made libraries/packages comprised of related classes and interfaces;

- Together, they insulate the Java programs from the underlying/host hardware-based platform.



The Java Programming Language

- Java programs are distributed as instructions for a virtual machine, making them **platform-independent** → so java programs can be run on any platform (Linux, OS, Windows, Android, etc.) once the Java Runtime Environment has been installed.

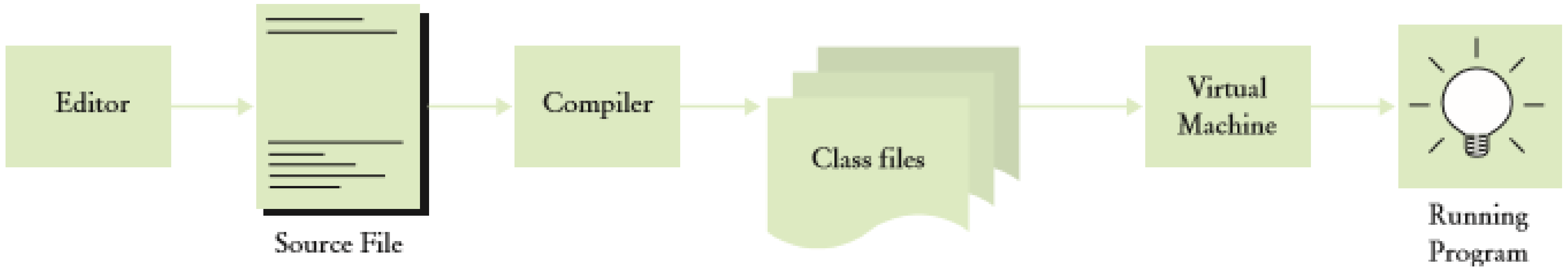


```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println("Hello World!");
        }
}
```

Java is designed to be:
-Simple
-object- oriented,
-safe/secure,
-multithreaded,
-distributed,
-dynamic,
-architecture-neutral,
- robust,
- high-performance and
-portable;

The 5 Phases of Java Programs

- There are 5 phases through which java programs typically go: edit, compile, load, verify and execute.
- The 1st **phase (edit)** takes place when we type and save the file;
- The 2nd **phase (compile)** takes place when we specify the compile command;
- The 3rd **(load)**, 4th **(verify)**, and 5th **(execute) phases** take place when we specify the command to run the program.



The 5 Phases of Java Programs

1. **PHASE 1 – EDIT:** writing and/or editing your program in a text editor, then saving it/store it on disk as a file that ends with .java;

- The java program that is written is typically referred to as the **java source code**;
- The file where the code is written must be saved as a java file, that is it must be saved with **.java extension**.
- For example, you can save the file as MyFirstProgram.java, or HelloWorldApp.java, or Welcome.java, etc.
- **Two types of editors** can be used: standalone, such as Notepad, or those that are part of an IDE (integrated development environment), such as Eclipse or Netbeans.

The 5 Phases of Java Programs

2. **PHASE 2 – COMPILE:** translates the java source code into bytecodes that are stored in a file that ends with .class which is generated automatically; these bytecodes are instructions for the JVM.

- The Java compiler is executed by using the **javac command**;
- You can do this explicitly in the command line:
  - if you have saved your source code as Welcome.java in the previous phase, then you must use javac Welcome.java command
  - if you have saved your source code as MyProgram.java in the previous phase, then you must use javac MyProgram.java command
- Or you can use shortcuts with some editors: for instance, in Textpad you can use CTRL + 1, or the compile button
- Note that your program will compile only if your source code contains **no syntax errors**.
- Also, every time you make a change to the source code, you must **re-compile** the program for that change to be translated into bytecode.

The 5 Phases of Java Programs

- The **last three phases** take place when we call the JVM to run the program;
- To run the program we use the **java command**
- explicitly in the command window
    - if you have saved your source code as Welcome.java and compiled it using javac Welcome.java command in the previous 2 phases, then you must use java Welcome command to run the program
- You can use shortcuts; for instance, in Textpad: CTRL+2 (or the run button).

3. **PHASE 3 – Load the program into memory:** the JVM transfers the bytecodes from the .class file into the primary memory;

- The class files can be loaded from a disk, or over a network.

The 5 Phases of Java Programs

4. **PHASE 4 – Bytecode verification:** the JVM checks that the bytecodes are valid and do not violate any security restrictions.
5. **PHASE 5 – Execution:** the JVM executes the instructions specified in the bytecodes;

- **NOTE:** Today's JVM uses a combination of interpretation and just-in-time (JIT) compilation to make the execution process faster – the JVM analyses the bytecodes as they are interpreted searching for parts of bytecodes that execute frequently.
- **NOTE:** Although the code may compile fine (i.e. there are no syntax errors), the program may encounter runtime errors or it may not work the way we want it to work (typically associated with logical errors) → software design and implementation/programming is an iterative process that typically requires to go through all these phases many times.

The General Syntax of a Java Program

The keyword ***class***

The <u>name/identifier </u>of the class

The <u>signature of the main method</u> which is the starting point of any java application

```
public class MyFirstProgram { //where the class begins

    public static void main(String[] arguments) { //where the main method begins

        System.out.println("This is my first attempt at Java!"); /* instruction to print
to      the screen the message within the quotation marks */

    } //where the main method ends

} //where the class ends
```

Opening and closing curly braces for the main method

Opening and closing curly braces for the class

Instruction given to the JVM to output the message <u>This is my first attempt at Java!</u>

The General Syntax of a Java Program

- The **class** is the building block in Java;
- In any java source file you can declare as many classes, but **only one class can be declared as public**, and, in that case, the file name must match exactly the name of the public class;
- Every java application has a class that contains the **main method** (but not every class has the main method, as we will see later); when an application starts, the instructions in the main method are executed, so every application must have a main method in order to run;
- A class can contain more than one method (we will learn about it later); and each method contains a sequence **of instructions called statements**.
- Instructions can take various forms (as we will learn), but in our program the statement **System.out.println("This is my first attempt at Java!");** constitutes a method call which is denoted by the pair of parentheses;
- Every statement must end with a semicolon ;

Java Programs

- The keyword **class** is compulsory; it must be followed by the **name of the class** (something that we can choose, but it must respect certain rules);
- **NOTE:** Keywords are reserved for use by Java and are always spelled with all lowercase letters; they include (but not limited to) class, public, static, void, etc.
- The contents of a class (methods, variables, etc.) must be enclosed within a set of **{ }** → Each class must begin with an opening curly brace **{** and end with a closing curly brace **}**;
- Each method must also begin with an opening curly brace **{** and end with a closing curly brace **}**;
- **the main method is optional** but, if you want your application to run, you must have it in your program; it typically contains instructions to be executed by the JVM.
- Everything that goes between the double quotation marks is a **String** in Java

REMEMBER: Java is case sensitive—uppercase and lowercase letters are distinct

How To Name Your Classes

- By convention, the **name of the class should begin with a capital letter** and then capitalize the first letter of each word they include;

  Example: MyFirstProgram, WelcomeProgram, etc.

- **A class name** is an identifier – a name that is a series of characters consisting of letters, digits, and/or certain special characters ;

- Other identifiers are used to name variables and methods (as we will learn later);

RULES FOR IDENTIFIERS – if not respected, the code will not compile:

- is **not already a reserved word** in the Java language (such as **class**, **void**);
- the name has **no spaces** in it:

  myVar is a valid identifier, but my Var is not a valid identifier
- the name does not include operators such as +, ! and -: var1+var2 is not valid identifier
- the name can only start with either with a **letter**, an **underscore** (_), or a **dollar sign** ($):

  name, _name, $name, name1 are valid identifiers

  1name, +name, %name are not valid identifiers

Some Reserved words in Java

exit
break
void
if
case
end
system
double
do
else
(etc…)

Note:

- These reserved words cannot be redefined for other uses.
- So, they may not be used on their own as identifiers.

EXAMPLE:

exit is not a valid name because is reserved

exit1 is a valid name because exit is not on its own, is used together with the digit 1

The 5 Phases of Java Programs in Action

- Once you have installed the JDK and some text editor, there are several options for creating and running your programs:

OPTION 1 – using the command window:

Step1. Open a text editor such as notepad (all computers have it)

Step 2. Open a terminal window

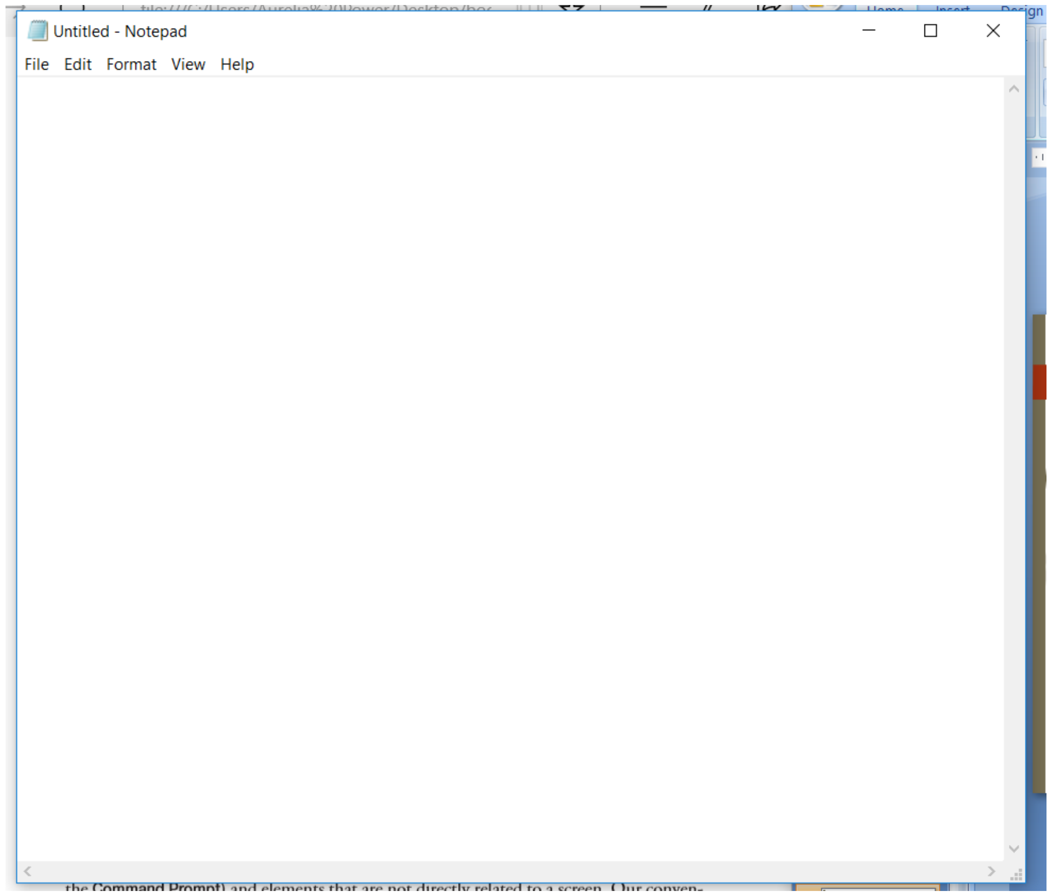
Step 3. Write the program and save it as java file

Step 4. Compile the program using the javac command

Step 5. Run the program using the java command

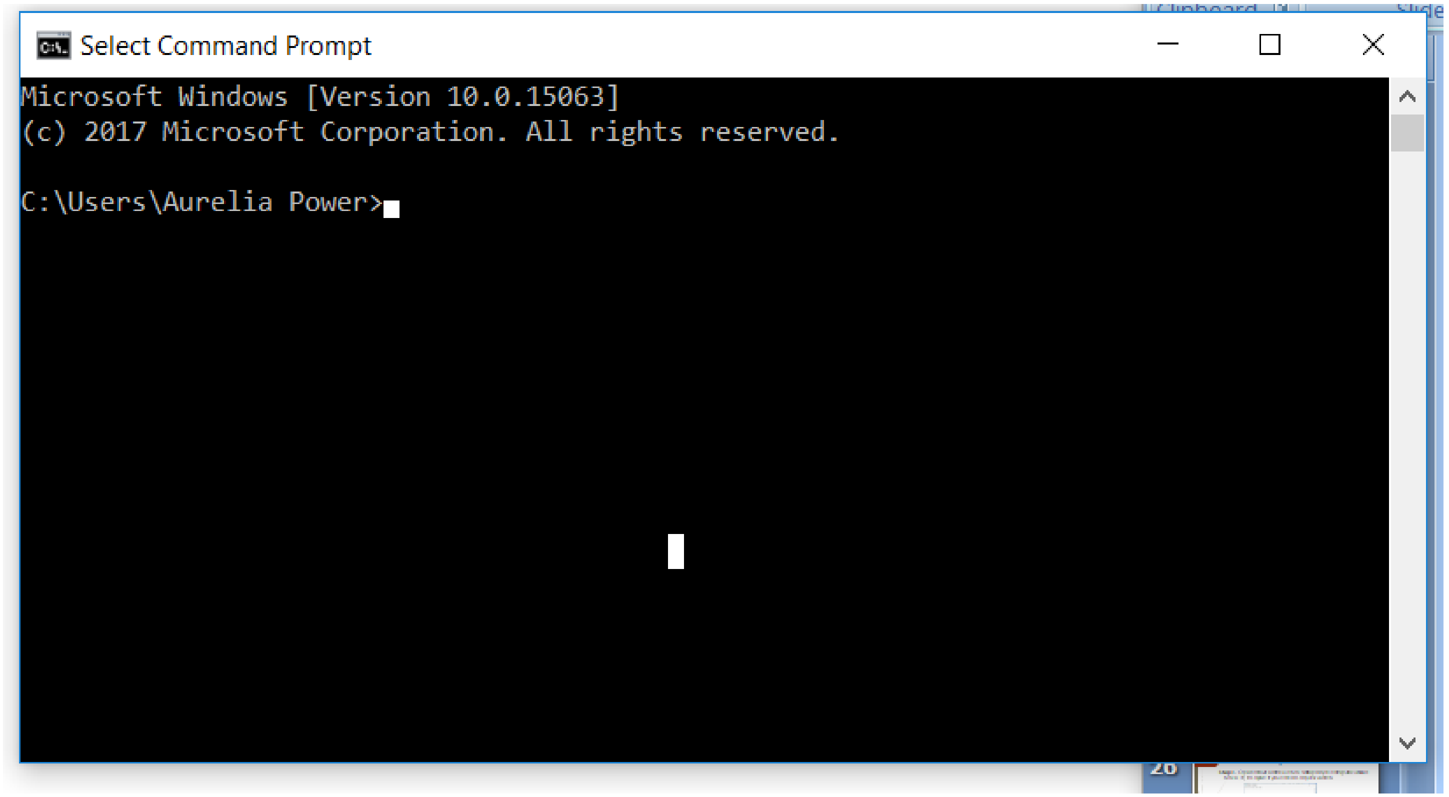
OPTION 1 – using the command window:

Step1. Open a text editor such as Notepad (all computers should have it) to open it you can do a quick search



OPTION 1 – using the command window:

Step 2. Open the command window

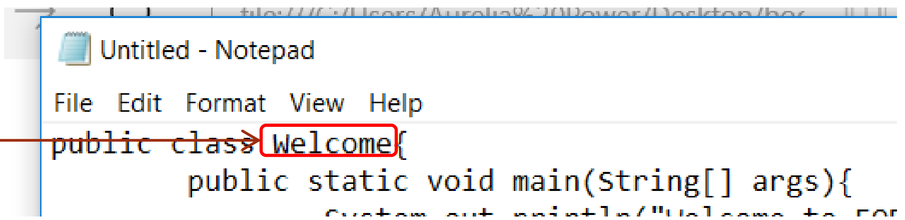


```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Aurelia Power>
```

OPTION 1 – using the command window:

Step 3. Write and save the program

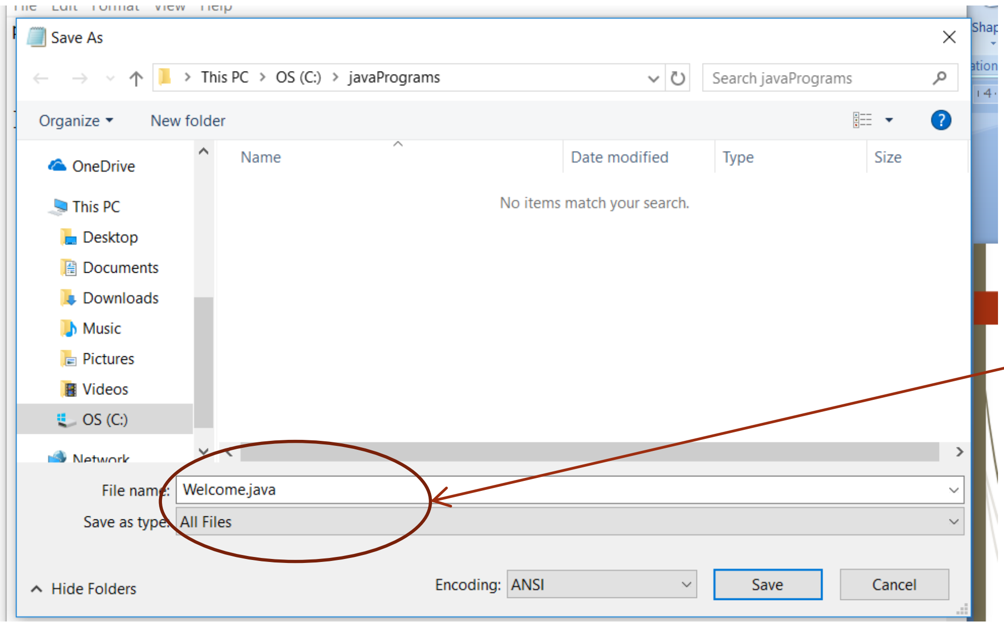
Name of the program/class



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

- **-** Save it using the name of the class as Welcome.java using CTRL+S command or the save as button
- **-** **Note:** when you save it you must make sure that the name of your program matches the name of the file (including the case, since java is case sensitive)
- **-** **Note:** also, when you save it, you must ensure that you save it as a java file, and not as a txt or some other type of file, by typing the name of the program followed by the .java extension, and then select save as All Files (see next slide).

Step 3. Write and save the program – ctd.



No items match your search.

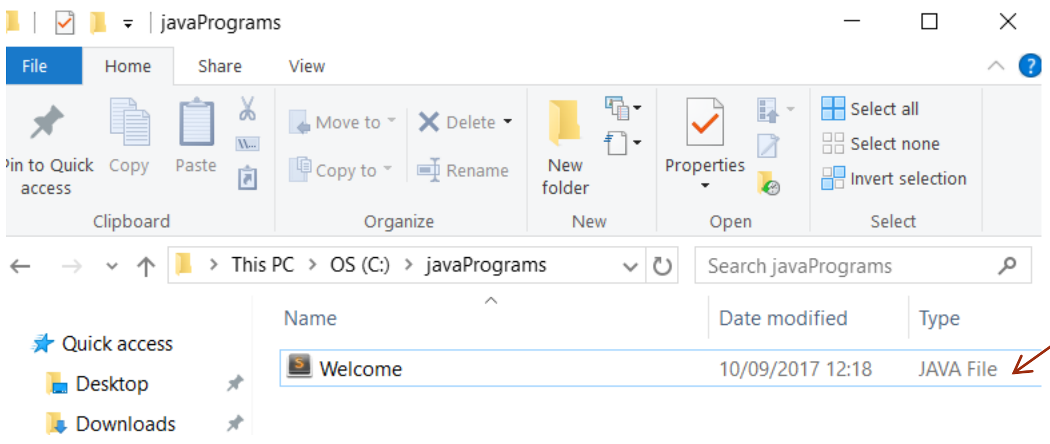
File name: Welcome.java

Save as type: All Files

Encoding: ANSI

Name of the program followed by .java extension, then select "All Files"

If saved correctly, a file with that name should be found in that directory



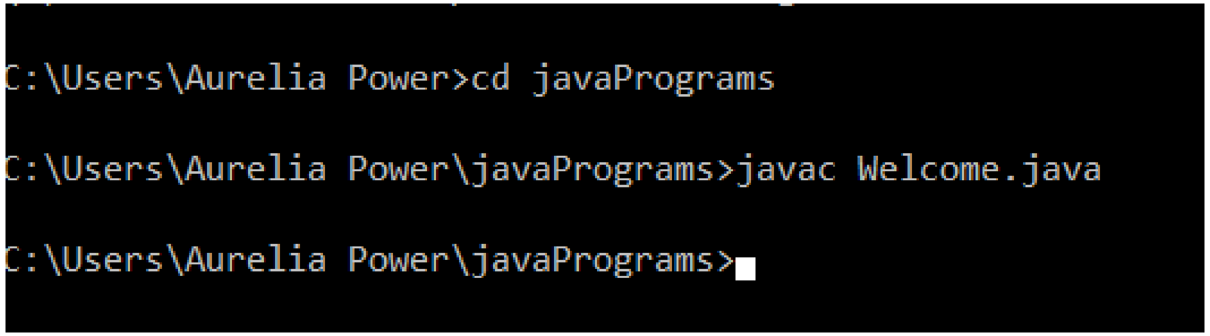
Name	Date modified	Type
Welcome	10/09/2017 12:18	JAVA File

A file with the same name as the program and the JAVA extension is found in that directory

OPTION 1 – using the command window:

Step 4. Compile the program

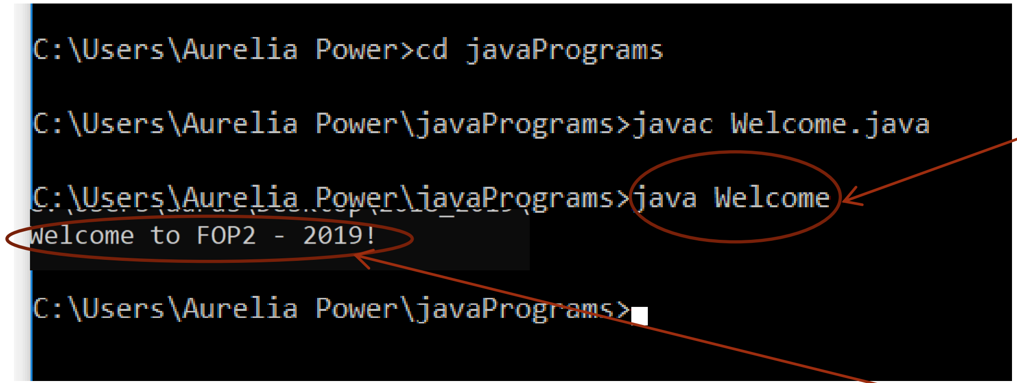
- To compile it correctly, you must select the directory where you saved the JAVA file using the cd command (which means change directory to…)
- Then use the javac command to compile the program;
- If there are no syntax errors, then another prompt will appear, and an associated CLASS file will be generated in the same directory as the JAVA file.



```
C:\Users\Aurelia Power>cd javaPrograms

C:\Users\Aurelia Power\javaPrograms>javac Welcome.java

C:\Users\Aurelia Power\javaPrograms>
```

OPTION 1 – using the command window:

Step 5. Run the program using the java command:



```
C:\Users\Aurelia Power>cd javaPrograms

C:\Users\Aurelia Power\javaPrograms>javac Welcome.java

C:\Users\Aurelia Power\javaPrograms>java Welcome
Welcome to FOP2 - 2019!

C:\Users\Aurelia Power\javaPrograms>
```

The command java Welcome that we typed in

The computer printed in the console the message "Welcome to FOP2 – 2019!" after we instructed the JVM to run the instructions contained in the program

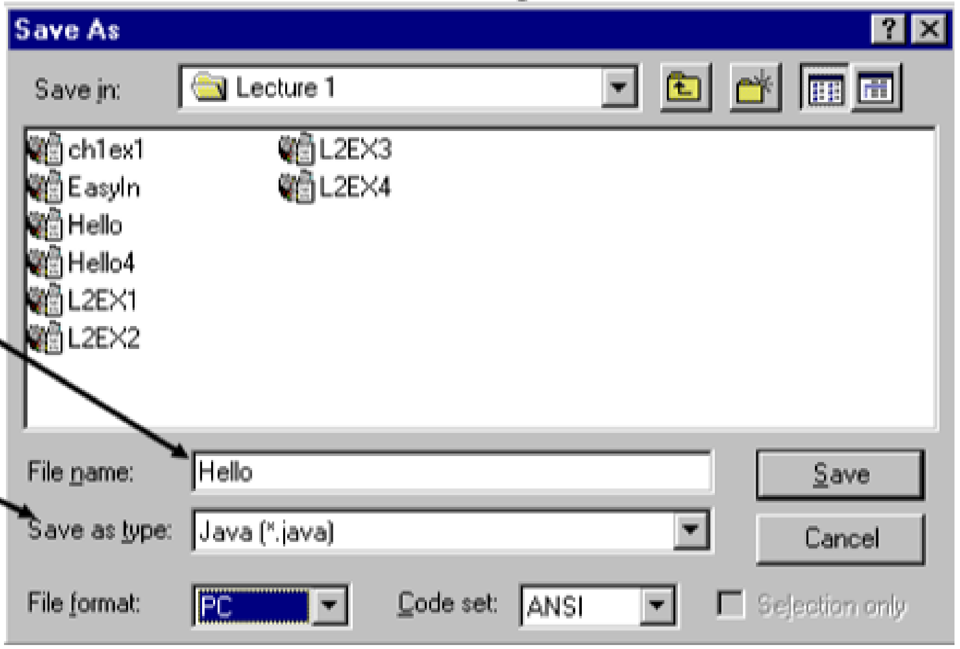
- Only if there are no runtime errors, the instructions inside the program will be executed and the program will output Welcome to FOP2 - 2019!

OPTION 2 – using the TextPad:

- Textpad to enter Java programs.
- You can execute this from Start/Programs/Textpad

To save the program

- Once you have created the program you should save it as <classname>.java
  In our last example program was saved as Hello.java
- Command is File/Save As
- Enter Program name in File Name
- Ensure Save as type is Java


Save As

Save in: Lecture 1

ch1ex1
EasyIn
Hello
Hello4
L2EX1
L2EX2
L2EX3
L2EX4

File name: Hello

Save as type: Java (*.java)

File format: PC

Code set: ANSI

Compile Program

- Command in Textpad is **Tools/Compile** or **Ctrl 1.**
- If the program compiles without error, a **.class** version will be created in the directory where your program is stored. For example if I compile the program Hello.java, Hello.class will be created in the directory.

Execute Program(Application)

- Command is **Tools/Run Java Application** or **Ctrl 2**.

Knowledge check

- What are the 5 phases of a Java program?
- What command do you use to compile a program?
- What command do you use to run a program?
- What is the correct way of declaring the main method?
- What does the main method should contain?
- How many classes can you declare in a java file?

- Which of the following will compile?

```
public Class MyFirstAttempt {
    /*more statements here*/ }
```

```
Public class MyFirstAttempt {
    /*more statements here*/ }
```

```
public class myFirstAttempt {
    /*more statements here*/ }
```

```
public class myFirstAttempt {
    /*more statements here*/
```

```
public class MyFirstAttempt {
    /*more statements here*/ }
```

- Will the following statements compile?

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

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

```
Systen.out.println(“Hello World!”);
```

```
System.out.println“hello world!”);
```

```
System.out.println(“hello world!”);
```

```
System.out.println(‘hello world!’);
```

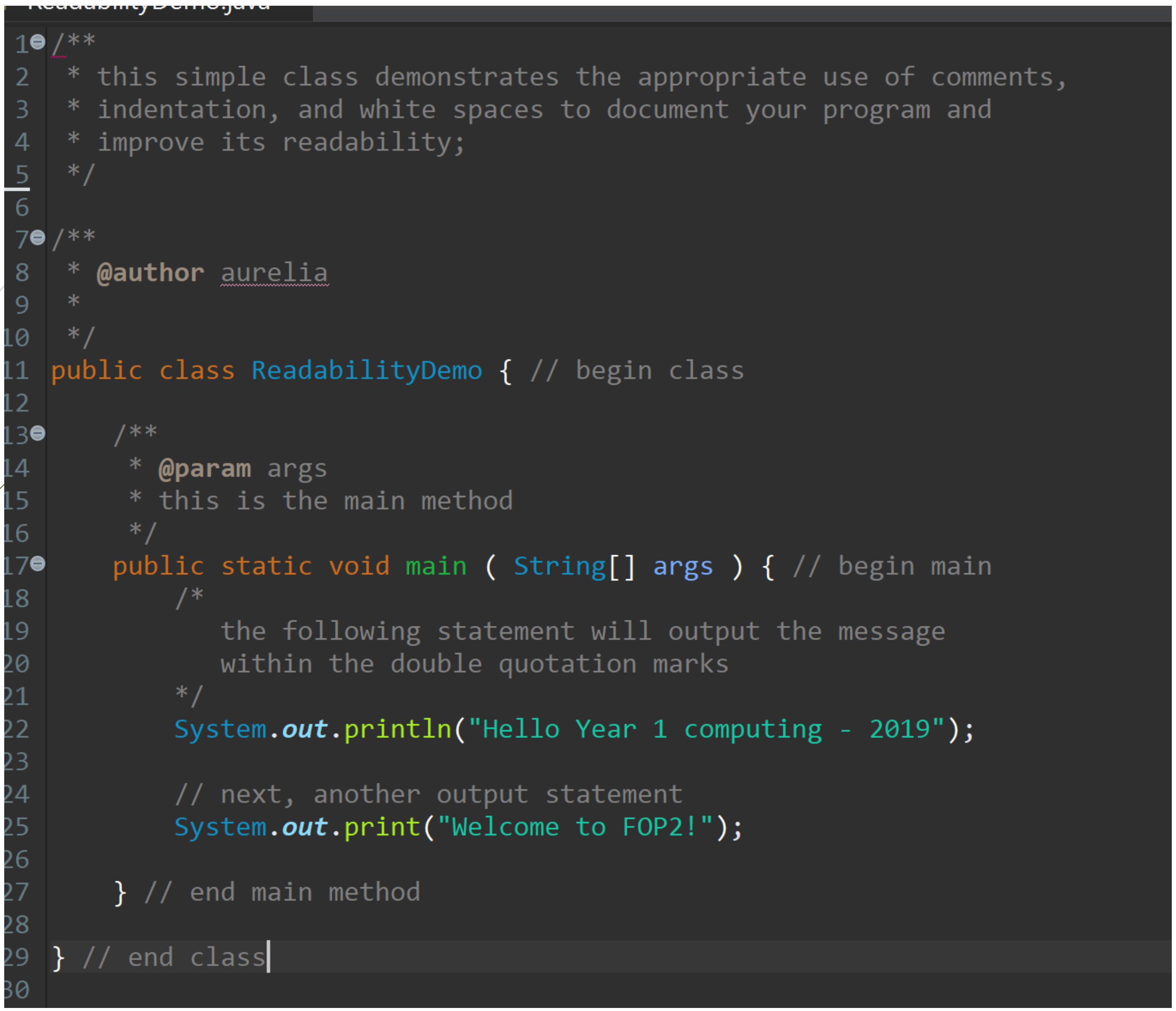
Improve Readability - Comments

- A java file can contain comments;
- **Comments** are used to document a program and increase readability; it is good practice to document your program so others can understand it.
- Comments are not instructions and are ignored by the compiler;
- There are several types of comments:

1. **Traditional comments** that can span over several lines: /* I am a traditional comment and I can be split over multiple lines */
2. **End-of-line comments** cannot span over several lines; it should be placed at the end of line: // I am an-end-of line comment; I cannot be split over multiple lines.
3. **Javadoc comments** that can also span over several lines: /** I am a javadoc comment and I can span over multiple lines; in industry, I am the preferred choice for documenting a program */

Improve Readability – Indentation, spaces and Blank Lines

- Tabs and space characters and blank lines make your code more readable
- They are known as **white spaces** – because the compiler ignores them:
- It is recommended that you indent each line of code according to the level to which it belongs:
  - The code immediately belonging to the class should be indented one tab
    - The code belonging immediately to methods found in the class should be indented with an additional tab
      - And so on…
- White spaces should be used also between operators, braces etc.
- Between blocks of code is recommended to use blank lines.



```java
/**
 * this simple class demonstrates the appropriate use of comments,
 * indentation, and white spaces to document your program and
 * improve its readability;
 */

/**
 * @author aurelia
 *
 */
public class ReadabilityDemo { // begin class

    /**
     * @param args
     * this is the main method
     */
    public static void main ( String[] args ) { // begin main
        /*
           the following statement will output the message
           within the double quotation marks
        */
        System.out.println("Hello Year 1 computing - 2019");

        // next, another output statement
        System.out.print("Welcome to FOP2!");

    } // end main method

} // end class

```

Java Programs – basic output

- **Strings** (series of characters); they must be enclosed within **double quotation marks**;

  "I love programming" – is a String

  "Einstein" – is a String

- But, unlike in Python, we have a dedicated datatype **single characters** which must be enclosed **within single quotation marks;**

  System.out.println('a'); System.out.println('&'); System.out.println('1');

  'A' or ' ' (an empty space) or '*' or '7' – are all single characters (in Java they are called **char**)

- **NOTE:** you cannot enclose more than one character within single quotes; if there are more than one characters that you need, then it must be a String:

  'AB' – will not compile

- **NOTE:** however, a string can contain only one character (a sequence of a single character…):

  "A" – will compile and is a String

  'A' – will compile but is a char

Java Programs – basic output

- We can also output **numerical values** and even the result **of mathematical calculations,** like in Python:

```
System.out.println(7);  // outputs 7
System.out.println(23.5); // outputs 23.5
System.out.println(-12); // outputs -12
System.out.println(7 + 3); // outputs 10
System.out.println(12 * 2); // outputs 24
System.out.println(-12 / 3); // outputs -4
```

OUTPUT:

```
7
23.5
-12
10
24
-4
```

- All the statements above, once outputting the respective values/results, will also place the cursor at the beginning of the next line → so each of these outputs will be printed on different lines because we used **println** method which moves the cursor to the beginning of the next line.

Java Programs – basic output

- But outputs can be printed on the same line using the method **print** instead of println:

```
System.out.print(7);  // outputs 7
System.out.print(23.5); // outputs 23.5
System.out.print(-12); // outputs -12
System.out.print(7 + 3); // outputs 10
System.out.print(12 * 2); // outputs 24
System.out.print(-12 / 3); // outputs -4
```

OUTPUT:

```
723.5-121024-4
```

- All the statements above, once outputting the respective values/results, will place the cursor right after the output and NOT at the beginning of the next line → so each of these outputs will be printed on the same line with no space in between them.

Java Programs – basic output

- So how can we print our outputs on the same line but separated by some spaces?
- There are 2 options:

1. We can use **printf** statements (we will learn about them later…)
2. we can use **String concatenation by which any data type can be joined to a String, making the whole thing a String;**

- String concatenation is achieved using the **+** sign which is also known as the **concatenation operator** when applied to joining a String to another String or to another value (unlike in Python…)

```
System.out.println( "Here are some numerical values: " + 7 + " " + 23.5 + " " + -12);
```

// it outputs **Here are some numerical values: 7 23.5 -12** all on the same line

NOTE: String concatenation is more complex

Java Programs – basic output

- **When you write your class, you cannot split a string over multiple lines** unless you use the concatenation operator:

```
System.out.println("Hello
world!");  X

System.out.println("Hello"
+ " world!");  √

System.out.println("Hello world!"); √
```

- **Your turn**: Write a simple class that uses concatenation, numerical values and mathematical addition to output the following:

  your brother's age is 4 and your sister's age is 3

  your age is your sibling's age put together: 7

Java Programs – basic output

- **NOTE** : the **+ sign** is also **the addition operator** when applied to numerical values;

Examples:

```
System.out.println(3+7); // 10
System.out.println(200 + 12); // 212
```

- You can also concatenate a String to the result of any mathematical operation:

Example: we want to print out the cost of 7 pairs of runners, each worth 100 euro

```
System.out.println("the total cost of 7 pairs of runners, each worth 3.00 euro, is: "
    + (3.00 * 7)); // we used the parentheses to force the compiler to carry out
    multiplication first before it concatenates the result to the message
```

// it outputs: **the total cost of 7 pairs of runners, each worth 3.00 euro, is: 21.00**

- **Your turn:** write a statement to print **3 + 7 makes 10**

Variables

- So far we have used literal values in our programs.
- But many times we need to store such values so we can use them later, and to do so we use variable;

Variables are used to store values;

A variable is a storage location in the computer's memory with a given name; each variable has name and holds a value.

- These **named locations** are called **variables** because their values are allowed to *vary* over the life of the program.
- **<u>Note</u>: variables in Java must be declared with a datatype before assign it a value, unlike in Python, because Java is a typed language, whereas Python uses dynamic typing (we will talk about it a bit more later…)**

Declaring Variables

- The procedure of creating **named locations/variables** in the computer's memory that will contain values while a program is running is known as **declaring a variable**
- **To create a variable in your program you must:**
- give that variable a **name** (of your choice);
- decide which **data type** best reflects the kind of values you wish to store in the variable.
- **data types** represent the types of values that reflect the kind of data in reality

EXAMPLES:

- ✓ *price* of a cinema ticket (e.g. 7.50 euro): **real number**
- ✓ *how many* tickets sold (10 tickets): **integer**

- In Java there are primitive data types that programmers can use - often referred to as the **scalar types**

What kind of data type can we use?? –Let's start with the primitive data types…

Java type	Allows for	Range of values
byte	very small integers	-128 to 127
short	small integers	-32768 to 32767
int	big integers	-2147483648 to 2147483647
long	very big integers	-9223372036854775808 to 9223372036854775807
float	real numbers	+/- 1.4 * 10^{-45} to 3.4 * 10^{38}
double	very big real numbers	+/- 4.9 * 10^{-324} to 1.8 * 10^{308}
char	characters	Unicode character set
boolean	true or false	not applicable

Data types

- **Literal values in Java have certain default data types:**

System.out.println(7); /* 7 as literal value is considered an int, not short, not byte */

System.out.println(7.5); /* 7.5 as literal value is considered a double, not a float*/

- **BUT when declare variable we can choose the datatype**

byte x = 7; /* x is considered a byte*/

float f = 7.5; /* f is considered a float */

- Because of these differences, initially, we will use only **int** to represent whole numbers and **double** to represent real numbers; otherwise, we may run into errors such as loss of precision…
- however, you still need to know about byte, short, and float.

Naming your variables

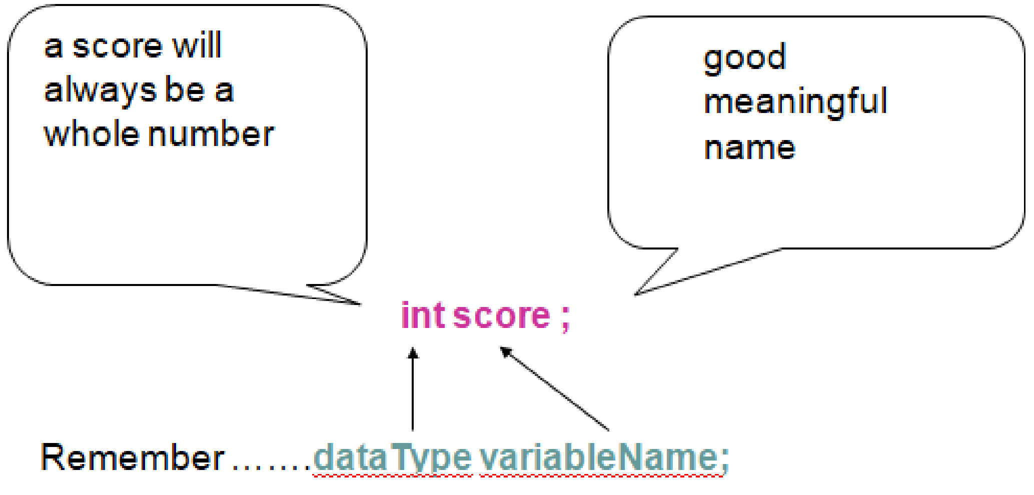
- When you declare a variable, you should pick a name that explains its purpose: numberOfStudents, kmPerHour, etc.
- The **same rules** as those used for naming classes apply, since names for variables are also identifiers:

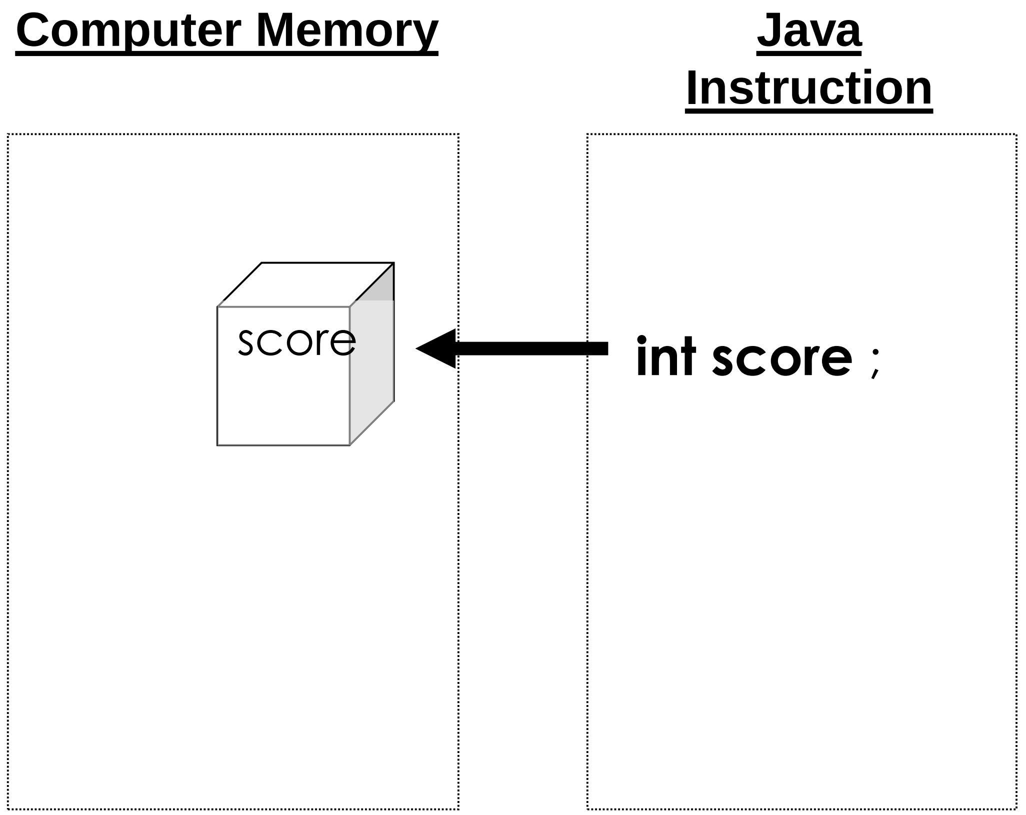
1. Variable names must start with a letter or the underscore or dollar character, and the remaining characters must be letters, numbers, or underscores, or dollars.
2. You cannot use other symbols such as ?, ! , +, - or % .
3. Spaces are not permitted inside names either. You can use uppercase letters to denote word boundaries, as in numberOfCoffees. This naming convention is called camel case because the uppercase letters in the middle of the name look like the humps of a camel.
4. Like in Python, variable names are case sensitive, that is, Student and student are different names/identifiers.
5. You cannot use reserved words such as double or class as names; these words are reserved exclusively for their special Java meanings.

NOTE: Not the same **naming convention** applies to classes and variable: classes should start with capital, variables should start with lower case.

Declaring variables

- Once we decide the best data type and once we decided the name **the variable is declared as follows:** **dataType variableName ;**


a score will always be a whole number
good meaningful name
int score ;
Remember…….dataType variableName;



Declaring more than one variable

- You can declare as many variables as you need.
- EXAMPLE: Assume that in addition to score, you also need to store a difficulty level of the game (A, B, or C) and the hits:

```
int score; // to hold score
int hits; // to hold hits
char level; // to hold difficulty level
```

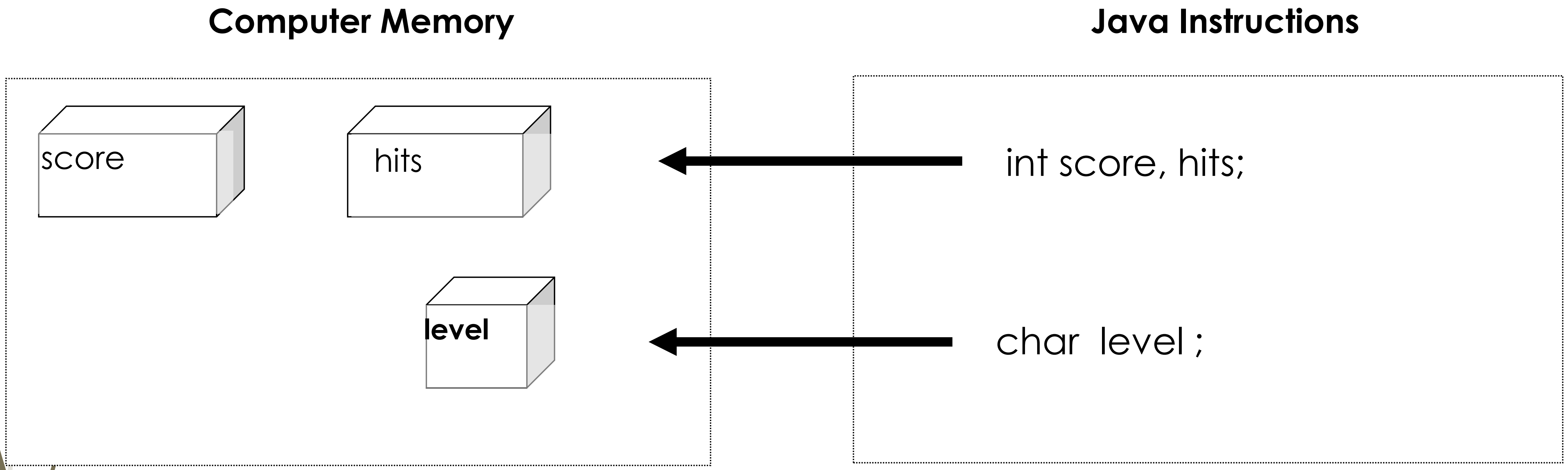
- Several variables can be declared on a ***single line*** if they are ***all of the same type***.

```
int score, hits; // both the same type
```

- However, once you declare a variable, you cannot declare again!!!

```
int score, hits; // compiles
int score; /* this line will not compile because we have re-declared score */
```

Let’s see again how the computer stores them in memory…



NOTE: **Every variable in a method must be first declared and initialised before it is used.**

Your turn…

1. Declare a variable called **age** that will store a person's age

2. Declare a variable called **noOfSeats** that will store the number of seats on a bus

3. Declare a variable called **noOfStudents** that will store the number of students in a lecture hall

4. Declare a variable called **bookPrice** that will store the price of a book

5. Declare a variable **groupNumber** that will store the group number that you are in for lab sessions

6. Declare a variable **grade** that will store the alpha grade you get in your assignments, e.g. A or B.

Assignments in Java

Assignments allow **values to be put** into **variables**.

The **equality symbol (=)** is used for assignments - known as **the assignment operator**.

Simple assignments take the following form (assume that they have been declared already):

variableName = value;

```
score = 0;

hits = 0;

level = 'A';
```

REMEMBER: **Every variable in a method must be first declared and initialised before it is used.**

Initialising variables

- ✓ You may **combine** the **assignment statement** with a **variable declaration** to put an initial value into a variable as follows:

```
int score = 0;
```

- ✓ Note, the following code will **not** compile in Java; why???

```
int score = 2.5 ;
```

This will not compile because 2.5 is a **double** value, not an **int**.

- ✓ What about this???

```
double score = 3;
```

It compiles because double values also include integer values, but not vice versa.

So the declared datatype on the left hand side must match or be inclusive of the data type that is assigned on the right hand side.

- ✓ When assigning a value to a character variable, you must **enclose the value in single quotes**. For example, if we want to set the initial value of a grade variable to A, we write the following assignment statement:

```
char grade = 'A';
```

Using the variable

- Once the declaration is made with a legal name (identifier), operations can be performed on the variable;

```
int age;
age  =  21;
System.out.print(age);
```

This command displays the value of age to the screen

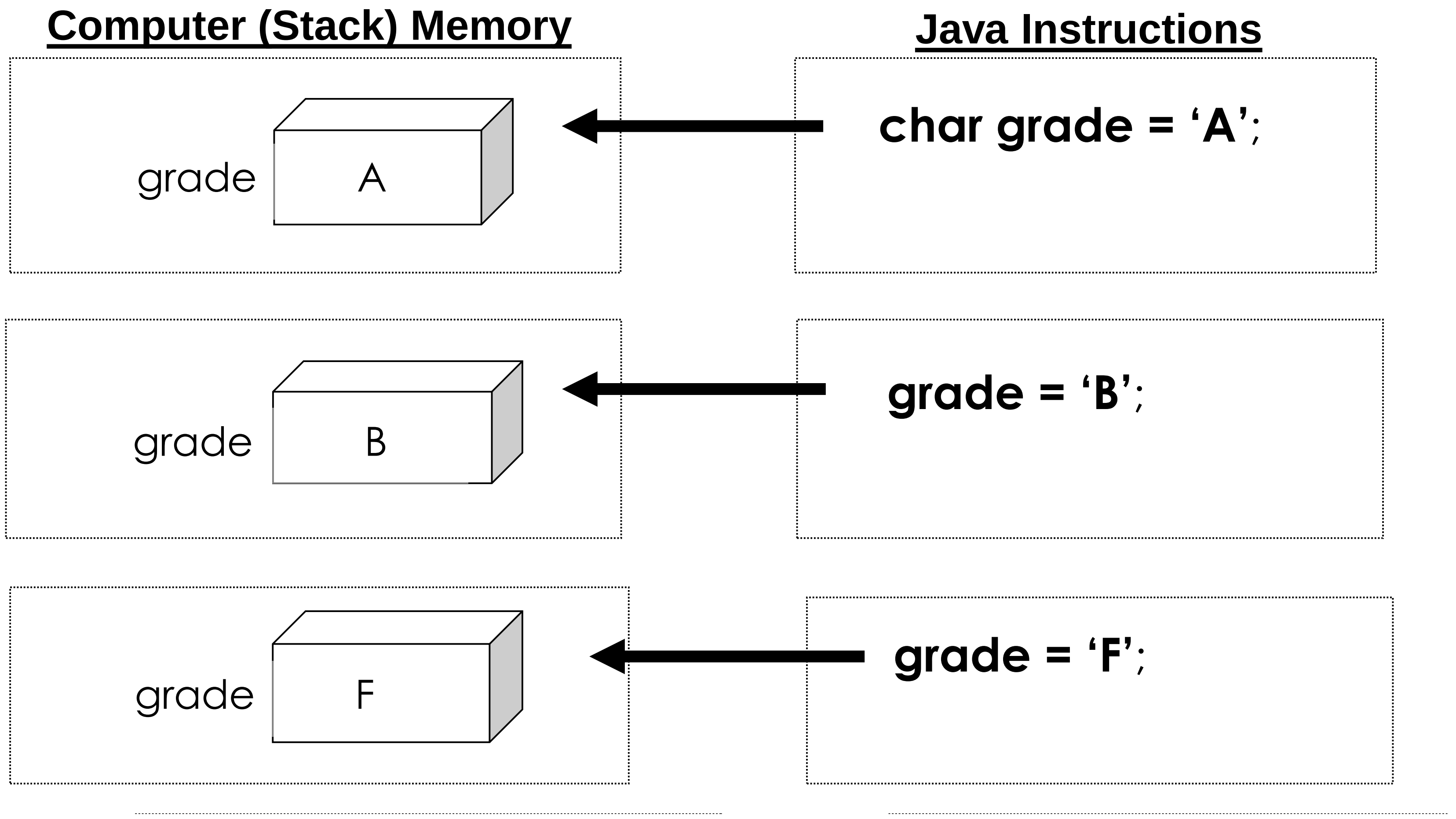
Which is … 21 (an integer value)

```
int ticketsSold = 20;
double pricePerTicket = 7.2;
System.out.print("Total cost = " + (ticketsSold * pricePerTicket));
// the output statement will print Total cost = 144
```

Re-assigning different values to variables

- Remember: you must declare a variable only once.
- But you can then re-assign different values to it as many times as you need.

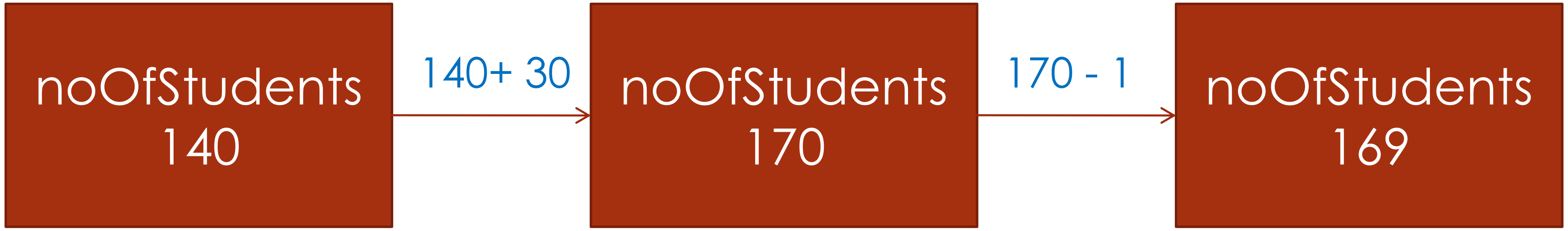
```
char grade = 'A';

grade = 'B';

grade = 'F';
```



Re-assigning values to variables and the effect on memory – another example…

```
int noOfStudents = 140;
//then I add 30 students
noOfStudents = noOfStudents + 30;
//then 1 student leaves
noOfStudents = noOfStudents - 1;
```

- The variable noOfStudents initially stores 140;
- Then to the old value of 140 is added 30, and the old value is destroyed and replaced by 170; now the variable noOfStudents stores 170;
- Finally, from the value of 170 is subtracted 1, and now the old value is destroyed and the noOfStudents takes the value of 169;



Creating Constants

Constants are data items **whose values *do not change***. For example:

- the maximum score in an exam (100);
- the number of hours in a day (24);
- the mathematical value of π (3.1417).

Constants are declared much like variables except:

- they are preceded by the keyword **final**
- they are always initialised to their fixed value.

```
final int HOURS = 24;
```

- You cannot re-assign values to constants;

```
HOURS = 23; // will NOT compile
```

Your turn…

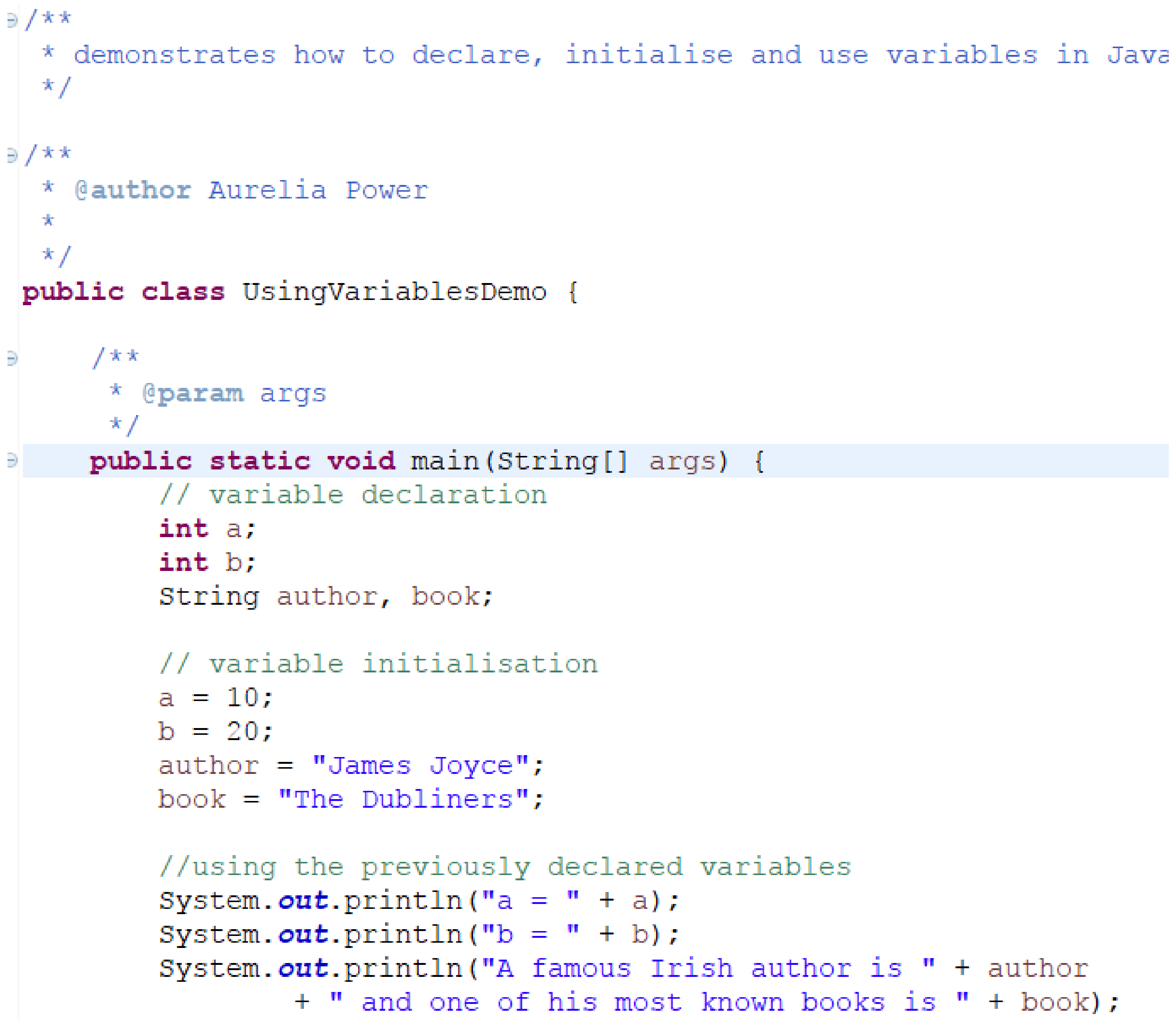
- Explain which, if any, of the following would result in a compiler error:

```
int x = 75.5;
String string = “string”;
char grade = A;
String  a = ‘a’;
double  value = 27.7;
```

- Do the following statements compile?

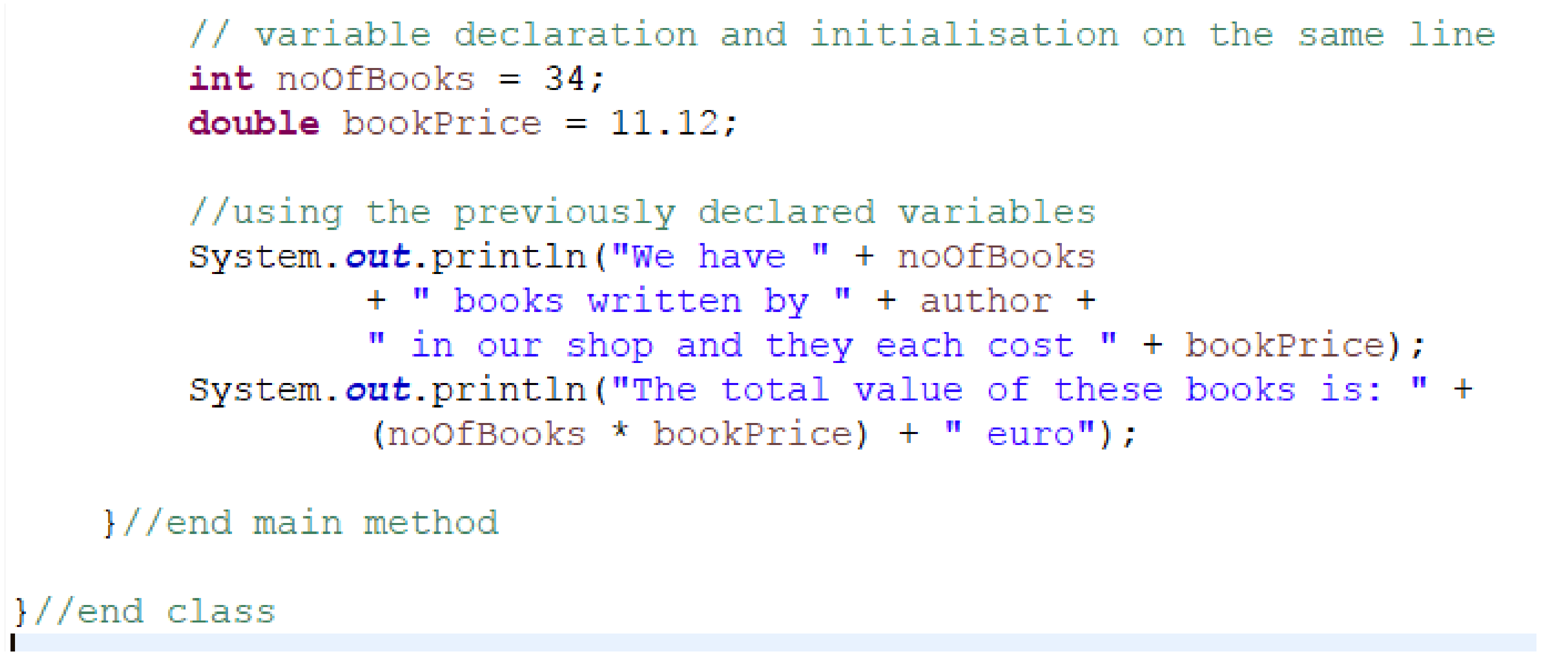
```
int x, y;
String s = “ ”;
int h = 10;
final int SIZE = 8.3;
char c = ‘hello’;
```

Sample Program (continued on the next slide)



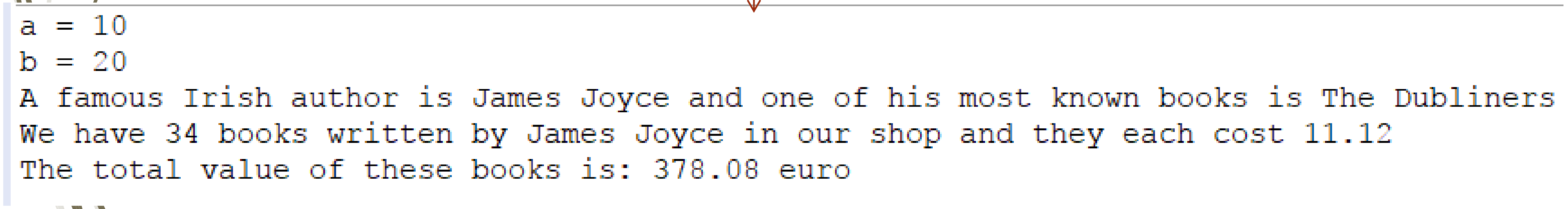
```
/**
 * demonstrates how to declare, initialise and use variables in Java
 */

/**
 * @author Aurelia Power
 *
 */
public class UsingVariablesDemo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // variable declaration
        int a;
        int b;
        String author, book;

        // variable initialisation
        a = 10;
        b = 20;
        author = "James Joyce";
        book = "The Dubliners";

        //using the previously declared variables
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("A famous Irish author is " + author
                + " and one of his most known books is " + book);
```

Sample Program (continued from the previous slide)



```java
        // variable declaration and initialisation on the same line
        int noOfBooks = 34;
        double bookPrice = 11.12;

        //using the previously declared variables
        System.out.println("We have " + noOfBooks
                + " books written by " + author +
                " in our shop and they each cost " + bookPrice);
        System.out.println("The total value of these books is: " +
                (noOfBooks * bookPrice) + " euro");

    }//end main method

}//end class
```

So when we run the whole program, we get the output below



```
a = 10
b = 20
A famous Irish author is James Joyce and one of his most known books is The Dubliners
We have 34 books written by James Joyce in our shop and they each cost 11.12
The total value of these books is: 378.08 euro
```

Your turn …

- Declare a variable suitable for holding the number of bottles in a case.
- What is wrong with the following variable declaration?

```
int ouncesperliter = 28.35
```

- Declare and initialize two variables, unitPrice and quantity, to contain the unit price of a single bottle and the number of bottles purchased. Use reasonable initial values.
- What is wrong with these statements?

```
double canVolume = 0.355; /*Liters in a 12-ounce can//
final int a = 3; a = 7;
```

- What are the values of the following variables after executing all the code?

```
int a = 9; int j = 10; double d = 7.7;
a = a+ 20; j = 2; d = d -10; a = j + 7;
```

- **<u>HOMEWORK</u>**: install the jdk and then the textpad on you machines and run all the examples and exercises from the lecture; also do the questions 1 sheet.