

# Software and Programming 1

## Lab 1: Introduction to Java: HelloWorld and InterestCalculator

# Module Information



## Lectures:

Afternoon 2pm (surnames A-H), 3.30pm (surnames I-Z)

Birkbeck Main Building, [Malet Street](#) MAL B04

Lecturer: [Carsten Fuhs](#)

Evening 6pm (surnames A-L), 7.30pm (surnames M-Z)

Birkbeck Main Building, [Malet Street](#) MAL B20

Lecturer: [Roman Kontchakov](#)

## Lab Sessions:

Afternoon 2pm (I-Z), 3.30pm (A-H) Main Building, [Malet Street](#) MAL 109

Evening 6pm (K-Z), 7.30pm (A-J) Main Building, [Malet Street](#) MAL 414/415

[Tobi Brodie](#), [Ping Brennan](#)

Module materials:

<http://www.dcs.bbk.ac.uk/~roman/sp1/>

## Additional Tutorials

5-6pm Birkbeck Main Building, [Malet Street](#) MAL 109

# Module Information

Generally, each class is split into two 90 minute sessions and, as there is a large attendance the class is also split, so the lecture and lab session you attend will be one of the following below:

- Lectures 2pm/3.30pm and 6pm/7.30pm,
- Lab sessions 3.30pm/2pm and 7.30pm/6pm

Attendance is compulsory for both Lectures and Labs, and a register is maintained.

Note: Lab sessions are designed to reinforce the material covered in the previous week's lecture, so there is no difference in which order you attend the lecture and lab session.

# Lab Session 1: Objectives

## Introduction to Java basics and the Blue J IDE

As this is the first week, we are not following up from a lecture, so we can spend this session familiarising ourselves with the syntax, coding conventions and data types of Java and look at the tools we will be using for programming on the module.

- Basic Java syntax, rules and coding conventions
- Java primitive data types
- Commenting
- [Blue J](#) – Application for coding in Java (free, cross-platform)
- Basics of methods and the terminal window output

In order to do this we will end the lab session by creating two programs:

**HelloWorld & InterestCalculator**

# Java Syntax

## syntax rules and coding conventions



### Rules (breaking a rule results in a compile-time error):

- Every variable declaration and assignment statement in a Java program must be terminated with a semicolon (;)
- Identifiers (variable, method and class names) are case-sensitive
- An identifier is a sequence of alpha characters, digits, underscores and \$ that begins with an alpha character
- Reserved words cannot be used for identifiers  
(see [https://docs.oracle.com/javase/tutorial/java/nutsandbolts/\\_keywords.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html))

### Coding Conventions (good practices, NOT rules):

- Variable, method and class names follow Camel Case: `studentNumber`
- Class names begin with a capital letter
- Method names begin with a lowercase letter
- Constants are named in CAPITALS
- Descriptive names are used for variables, not abbreviations
  - (e.g. `int speed = 70;` **not** `int s = 70;`)

# Java Data Types

Every variable in Java must be declared before it's first used. When declaring a variable in Java, we specify the ***data type*** along with the ***variable name*** (and optionally, an *initial value*).

For example, to declare a new variable containing a whole number we use the following syntax:

```
int studentNumber = 12311487;
```

This tells the compiler that there is a variable named `studentNumber`, which will hold an integer value. By using a single equals sign, we provide the initial assignment of the value `12311487` to this variable.

**Remember:** values from the right of an operator are assigned to the variable on the left.

Once a variable's data type is fixed, it will only accept values of that data type: `studentNumber` above cannot hold a value such as `2.5`

## Data Types (2)

Java supports 8 *primitive* data types:

- **byte** – integer values -128 to 127
- **short** – integer values -32,768 to 32,767
- **int** – integer values  $-2^{31}$  (-2,147,483,648) to  $2^{31}-1$  (2,147,483,647)
- **long** – integer values  $-2^{63}$  to  $2^{63}-1$
- **float** - 32-bit IEEE 754 floating-point numbers
- **double** - 64-bit IEEE 754 floating-point numbers
- **boolean** – values **true** or **false**
- **char** - 16-bit Unicode characters

In this module we will concentrate on the following primitive data types: **int** for whole numbers, **double** for floating-point numbers, **boolean** and **char**.

The 8 primitive data types are written in lower case (reserved words).

## Data Types (3)

In addition to the 8 primitive data types, Java provides support for character strings via the built-in `String` class.

The string data type begins with a capital letter and values are given within double quotations:

```
String message = "Hello World!";
```



# Java Comments

## Block Comments

Block comments are used to provide descriptions of classes, methods, data structures and algorithms.

```
/*  
* Here is a block comment.  
*/
```

*Coding Conventions:* Block comments may be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe. A block comment should be preceded by a blank line to set it apart from the rest of the code.

Block Comments can also be used as trailing comments:

```
if (a == 2)  
{  
    return 25;           /* special case */  
}
```

# Comments (2)

## End-Of-Line Comments

The `//` comment delimiter can comment out a complete line or only a partial line.

*Coding Conventions:* It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code.

```
if (number >= 0)
{
    // code for natural numbers
    ...
}
else
{
    return 0;           // number will not be in range
}
```

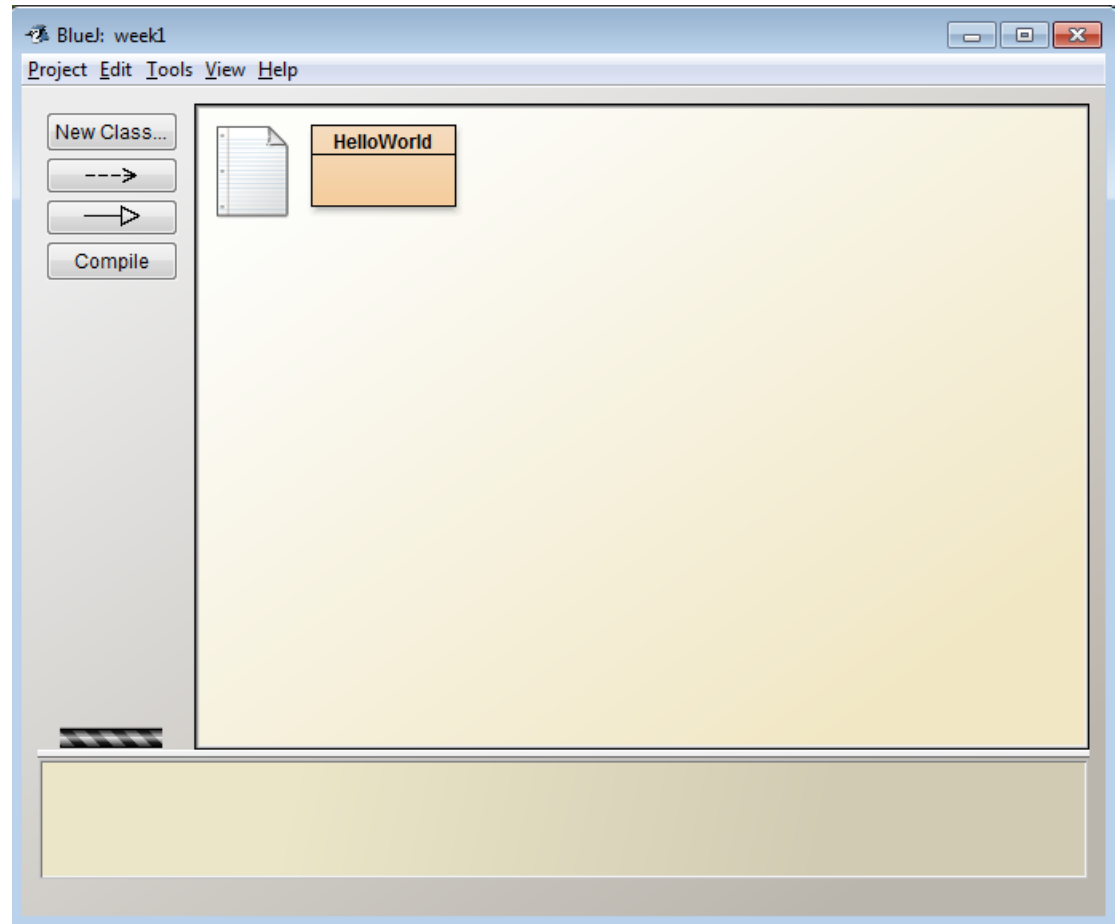
<http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-141999.html>

# Java Project in BlueJ

Name of project:  
**week1**

Name of class:

HelloWorld

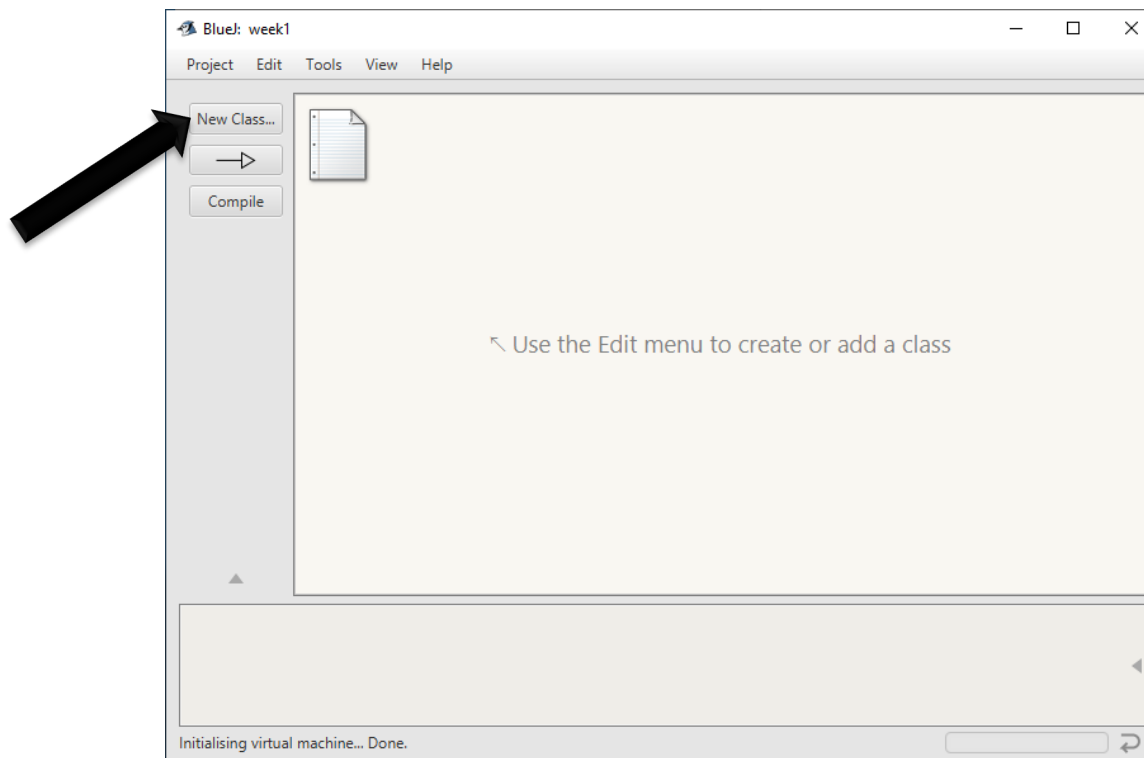


# Getting Started

- Launch BlueJ - begin with the **Start** icon in the lower left corner of the screen.
- Select the options in the order shown:  
**Start -> All Programs -> Programming Tools -> BlueJ**
- Create a new Project on your disk space.
  1. Select Project then followed by **New Project**.
  2. Select a directory in your disk space (in drive I on non-DCS computers) and a suitable name for your project, e.g. **week1**. After entering **week1** in the BlueJ window, a new BlueJ window will appear for the project **week1**.

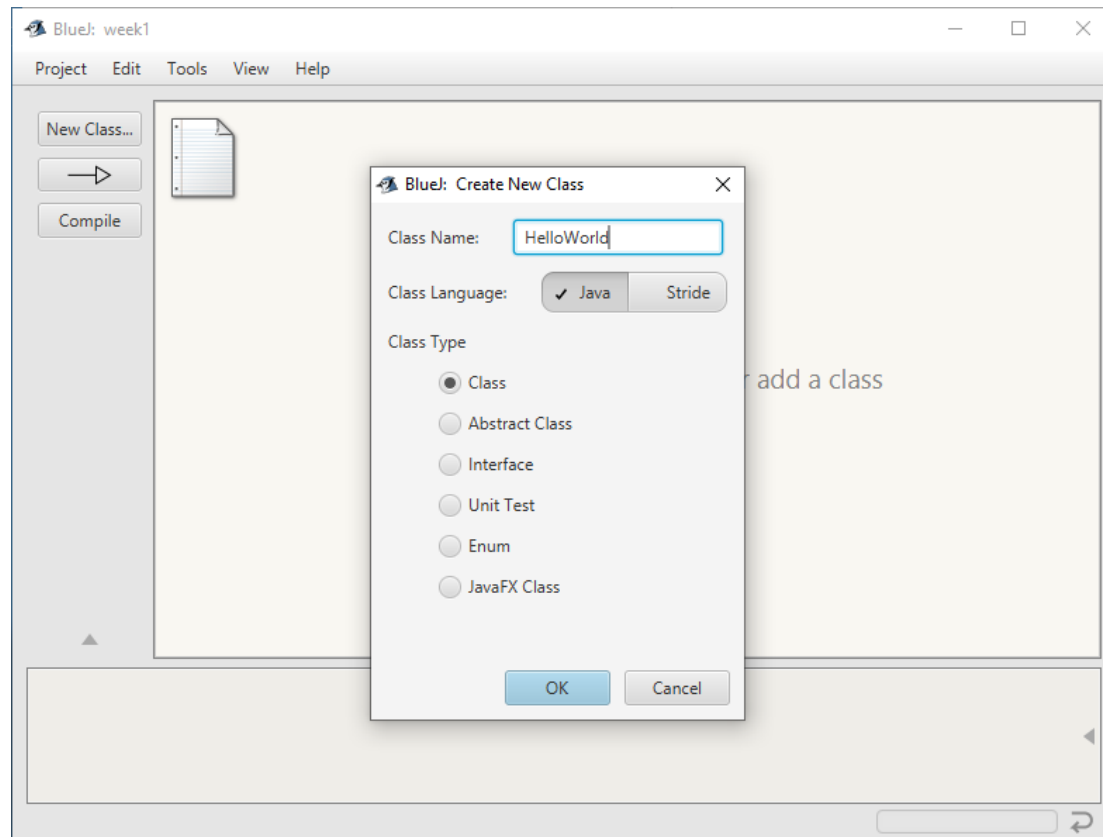
## Getting Started (2)

- Create a new class by clicking on button **New Class ...** in the new BlueJ window.



## Getting Started (3)

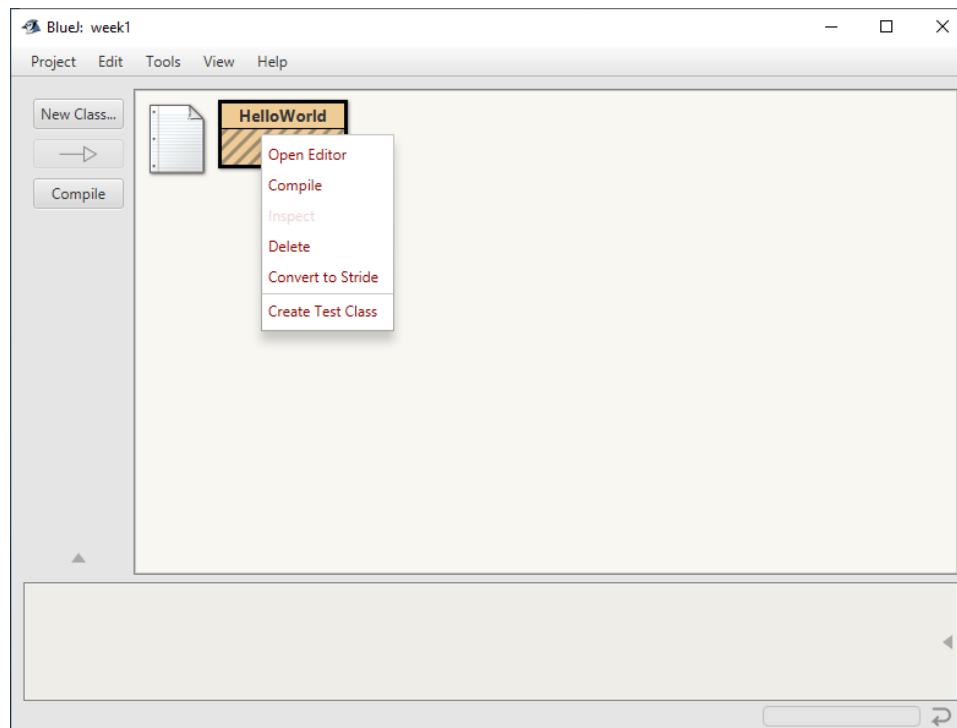
- Enter the name **HelloWorld** for the new class and click on **OK**.



# Exercise 1:

## Write your first program

- Move the mouse on top of the class icon with the name **HelloWorld**, right-click and select **Open Editor**.

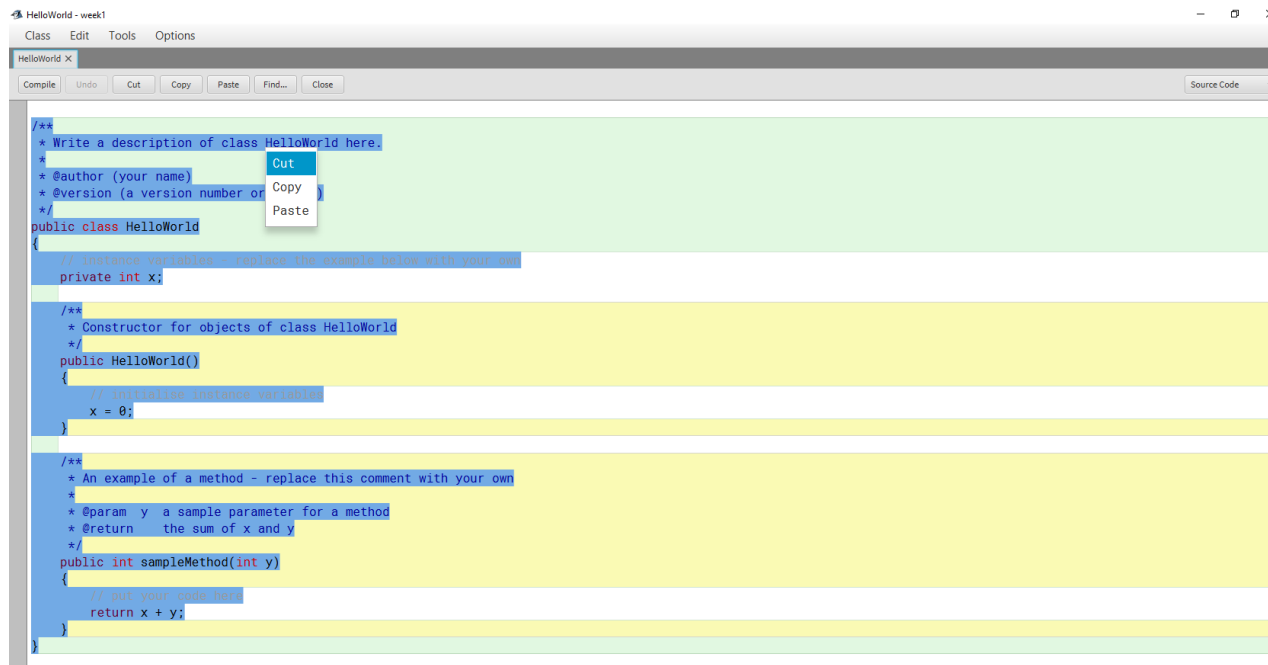


# Exercise 1 :

## Write your first program (2)

- Delete all the code in the template class and leave it empty for now.

This can be easily done by selecting all (control + A), using the right click and selecting cut.



```
/**
 * Write a description of class HelloWorld here.
 *
 * @author (your name)
 * @version (a version number of your code)
 */
public class HelloWorld
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class HelloWorld
     */
    public HelloWorld()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param y a sample parameter for a method
     * @return the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```



# Exercise 1:

## Write your first program (3)

- Writing your own code:
  1. Start by writing two keywords, **public class**.
  2. Write the name of the class, **HelloWorld**.
  3. First line of your code looks like: **public class** HelloWorld
  4. Any code that you might write next for the class HelloWorld must be put after the first line and it must be enclosed with braces (i.e. **{ }**).

```
public class HelloWorld
{
    /*
       all code must lie between the two braces that
       define the boundaries of the class
    */
}
```

# Exercise 1:

## Write your first method

- Steps in defining a method:
  1. First write **public static void**.
  2. Next write the method's name **main**.
  3. Followed by the method's parameters **String[] args** in brackets.
  4. Finally, followed by the method's boundaries (open/close braces { } ).Your code must look like:

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

    } // end of method
} // end of class
```

Note the indentations of the lines of code which make the code easier to read.

# Exercise 1:

## Write your first method (2)

5. Use the statement

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

within the method to make it print something in your terminal.

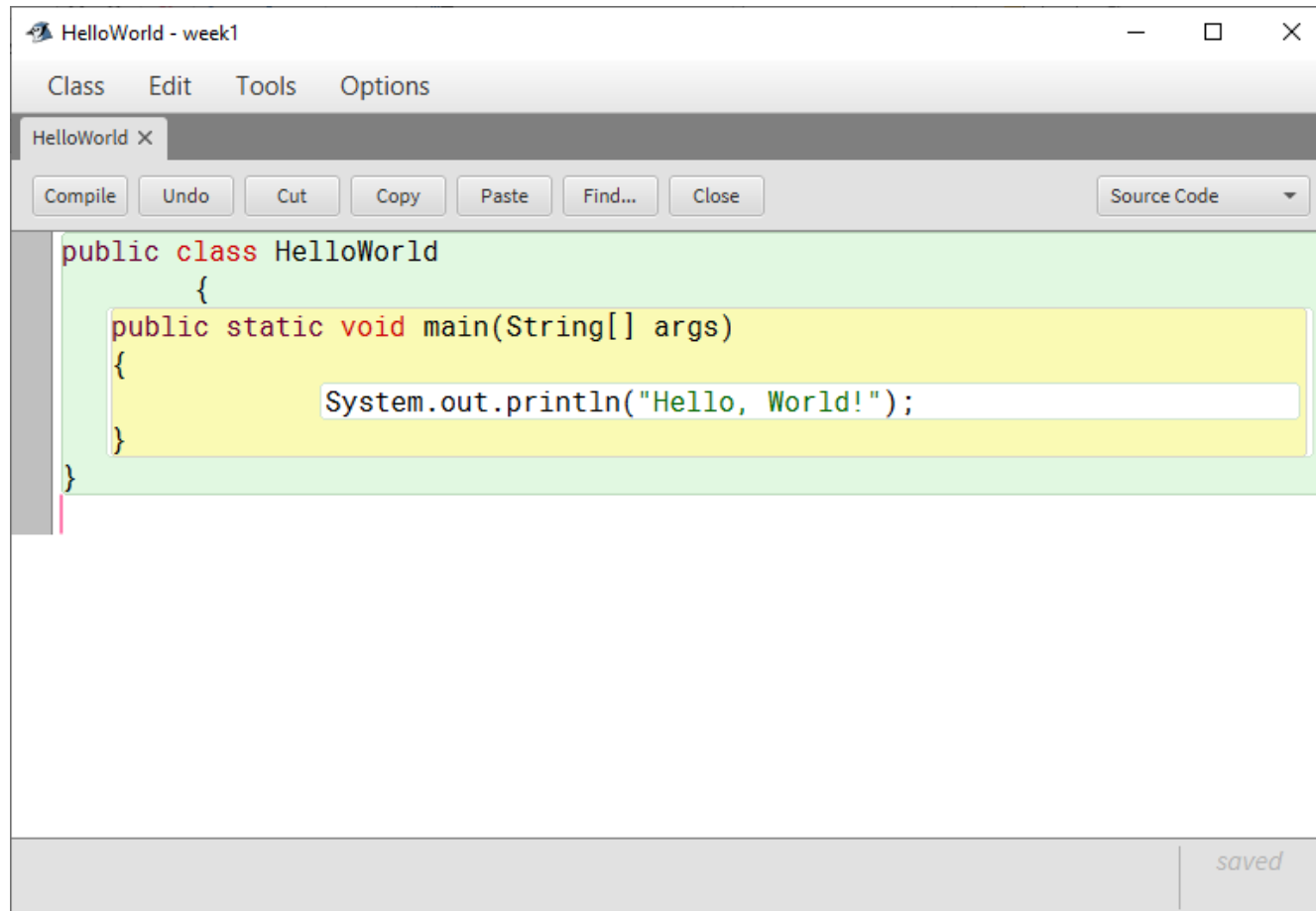
For example,

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

# Exercise 1:

## Write your first method (3)

6. Your code must look like this:



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

The screenshot shows a Java IDE window titled "HelloWorld - week1". The menu bar includes "Class", "Edit", "Tools", and "Options". The toolbar contains buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Close", and a "Source Code" dropdown. The code editor displays the following Java code:

# Exercise 1:

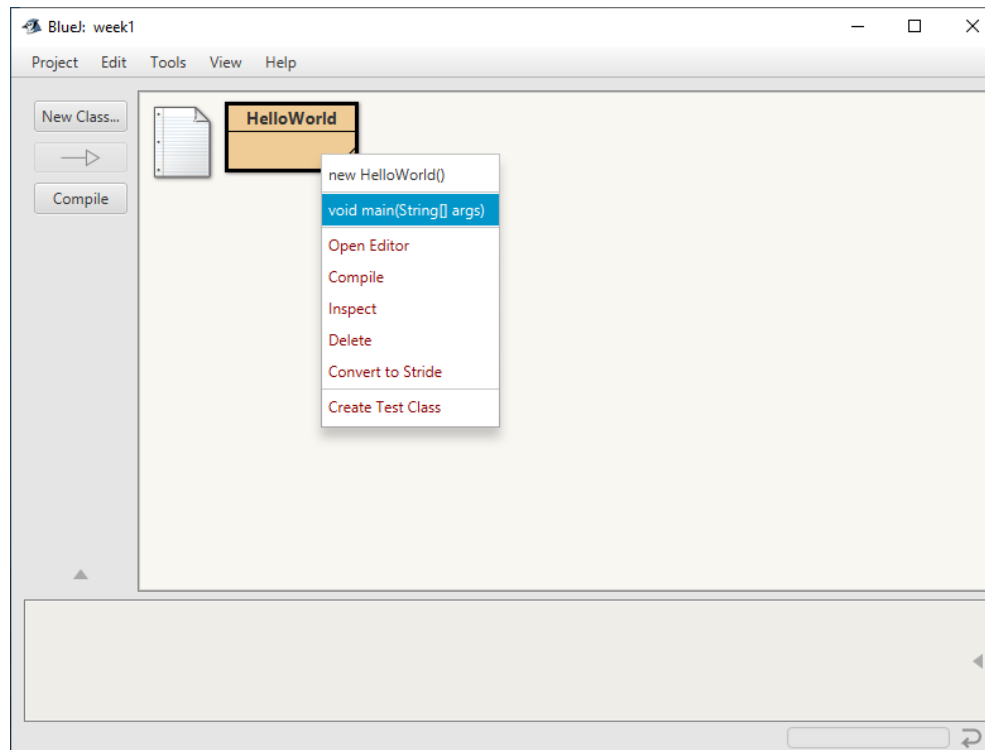
## Compiling your first class

- Click on the button **Compile**. The compiler will check your code for syntax errors and error messages (if any) are displayed at the bottom of the window.
- The final message should be one of the following.
  - Either **Class compiled – no syntax errors**
  - Or a compile-time error message.
- Important: after each modification of the code, **always compile the new code.**

# Exercise 1:

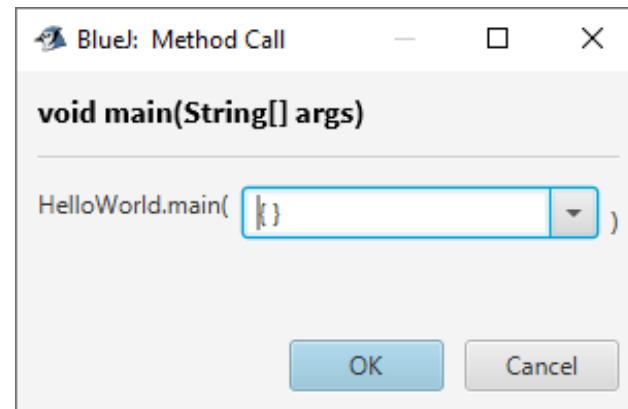
## Execute the method

- Close the Editor and return to the project's workspace.
- Move the mouse on top of the **HelloWorld** icon, right-click and invoke the method **main** by clicking on it.

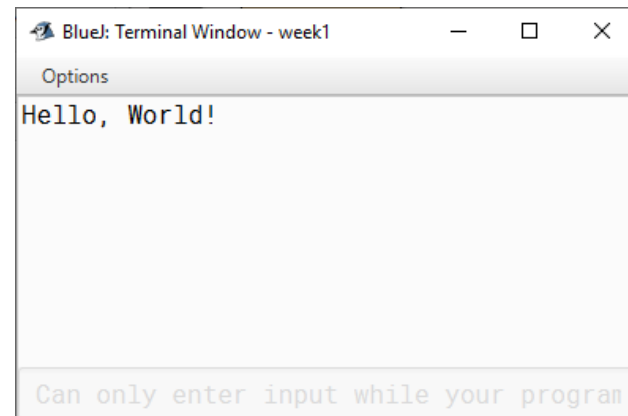


# Exercise 1: Execute the method (2)

- After selecting the main method, a window will appear and select **OK**.



- A terminal window will appear with the message:  
`Hello, World!`



# Note

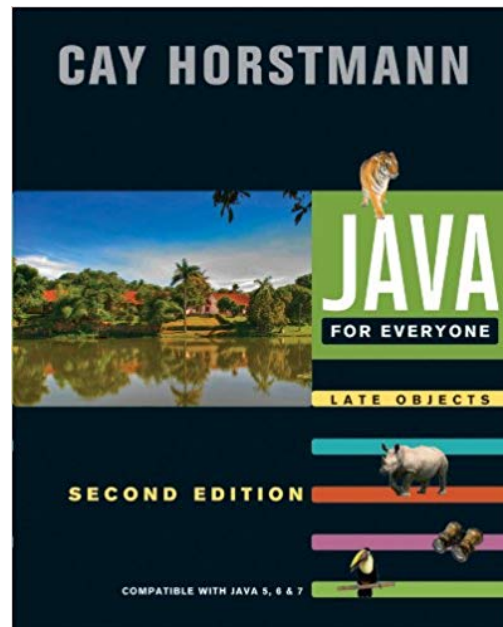
- The exercise just completed outlines the basic structure for creating Java programs, creating a method within the class, compiling and finally executing the method. Generally, the same structure will be applied for classes created during the sessions for the next few weeks.
- Exercise 2, which follows, requires you to again create a new class, define methods within the class, then compile and execute to see the result.



# Exercise 2: InterestCalculator

You put £10,000 into a bank account that earns 5% interest per year. What will the balance be after 3 years?

(Based on JFE, Section 1.7)



## Exercise 2: InterestCalculator (2)

- Initial balance: £10000
- Interest rate: 5% per year
- Interest earned after 1 year:  $10000 * 5 / 100 = 500$
- New balance after 1 year: initial amount + interest  
=  $10000 + 500$   
=  $10000 + (10000 * 0.05)$
- Balance after each subsequent year:  
= previous balance + interest on it

# Exercise 2: InterestCalculator (3)

## Pseudo code:

1. `initialBalance = 10000`
2. `Print "initial balance" + initialBalance`
3. `currentBalance = withInterestOn(initialBalance)`
4. `Print "year1" + currentBalance`
5. `currentBalance = withInterestOn(currentBalance)`
6. `Print "year2" + currentBalance`
- ...

Note: The code to calculate the balance is identical for lines 3 & 5 and will be for each successive year. A method to calculate the interest should be written in addition to the `main` method.

# Exercise 2: InterestCalculator (4)

```
public class InterestCalculator
{
    public static void main(String[] args)
    {
        // declare variable initialBalance
        System.out.println("The initial balance is £" +
                           initialBalance);

        // declare variable currentBalance
        currentBalance = withInterestOn (initialBalance);
        // print new balance

        /* repeat previous 2 lines (use currentBalance instead of
           initialBalance as argument for withInterestOn call)
           to calculate balance for 2 more years(see previous slide)
        */
    }
    public static double withInterestOn(double balance)
    {
        double interest = balance * 0.05;
        return balance + interest;
    }
}
```

# Exercise 2: InterestCalculator (5)

A closer look at the method to calculate interest:

```
public static double withInterestOn(double balance)
{
    double interest = balance * 0.05;
    return balance + interest;
}
```

When writing Java methods we must declare the data type of the return value (returned by the `return` statement) as well as the data types of parameters passed as arguments (`double balance` is a declaration of the variable that stores the value passed into the method from the method call)

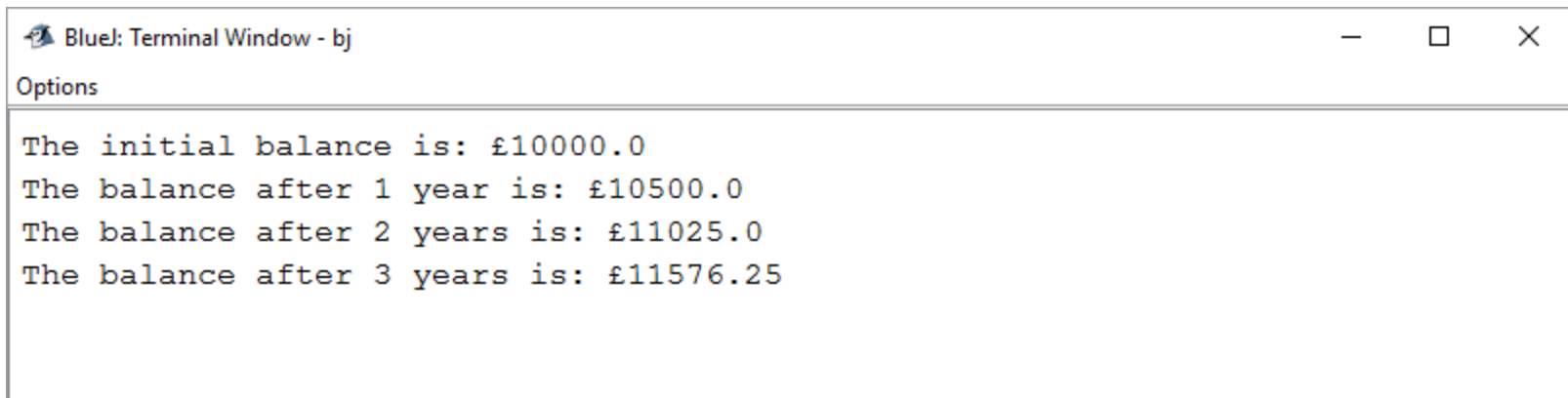
# Exercise 2:

## Compiling InterestCalculator

- Click on the button **Compile**. The compiler will check your code for syntax errors and error messages (if any) are displayed at the bottom of the window.
- The final message should be one of the following.
  - Either **Class compiled – no syntax errors**
  - Or an error message.
- Important: after each modification of the code, **always compile the new code**.

## Exercise 2: Execute the method

- Close the Editor and return to the project's workspace.
- Move the mouse on top of the **InterestCalculator** icon, right-click and invoke the method **main** by clicking on it.
- A window will appear and select **OK**.
- A terminal window will appear with output similar to below:



```
Blue: Terminal Window - bj
Options

The initial balance is: £10000.0
The balance after 1 year is: £10500.0
The balance after 2 years is: £11025.0
The balance after 3 years is: £11576.25
```

# Home Work

## Java for Everyone by C. Horstmann

Read Chapter 1, which is available online from

<http://vufind.lib.bbk.ac.uk/vufind/Record/566484>

and complete the following exercises:

- Exercise R1.8
- Exercise R1.9
- Exercise R1.15
- (\*extra) Exercise R1.18
- Exercise P1.3
- Exercise P1.5