

# Software and Programming I

## Lab 3: Loops and conditional statements

# Lab 3 Objectives

Understanding & Debugging loops.

Understanding which type of loop is correct for a particular program.

The correct use of conditional statements to get the correct results from common loop algorithms.

There are three exercises to complete this week.

**Note:** ***Powers** and **BalancedParentheses** classes are marked exercises and you are required to complete them and show the working program on or before the **13<sup>th</sup> February**.*

*Make sure you have backed up your work on a memory stick or similar.*

# Loops & Debugging

In order to remind ourselves how to work with loops and the correct syntax for Java, we will modify the **HelloWorld** program we just created in week 1 into **PrintHelloWorldWhile**:

```
public class PrintHelloWorldWhile {
    public static void main(String[] args) {
        int i = 0;
        while(i < 10) {
            System.out.println("Hello World!");
            i = i + 1;
        }
    }
}
```

Note: You will need to save this as a new class in BlueJ

# Loops & Debugging (2)

Before we run the code

```
int i = 0;
while (i < 10) {
    System.out.println("Hello World!");
    i = i + 1;
}
```

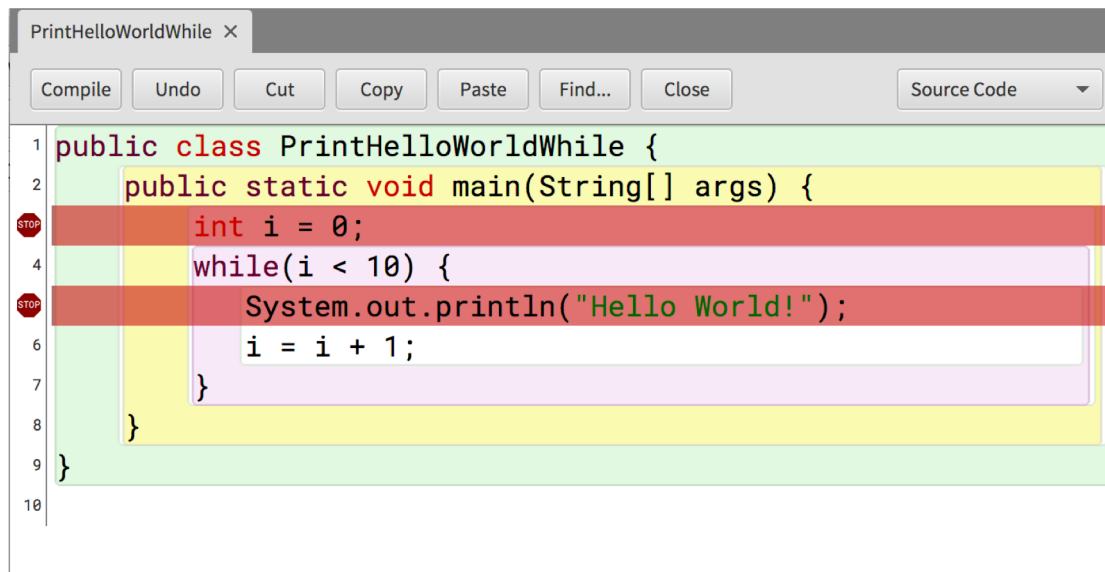
we should be able to work out how many times "Hello World!" is printed to the terminal window by quickly drawing out a table.

counter i (at start of loop)	i < 10	Output to terminal	i after the increment (i = i + 1)
0	true	Hello World!	1
1	true	Hello World!	2
2	true	Hello World!	3
3	true	Hello World!	4
4	true	Hello World!	5
5	true	Hello World!	6
6	true	Hello World!	7
7	true	Hello World!	8
8	true	Hello World!	9
9	true	Hello World!	10
10	false		

## Loops & Debugging (3)

By utilising the built-in Debugger in BlueJ we can make sure that the values of the counter are as expected and that our program will print "Hello World!" the correct number of times.

Add the following breakpoints to check the counter value against the loop table:



```
PrintHelloWorldWhile x
Compile Undo Cut Copy Paste Find... Close Source Code
1 public class PrintHelloWorldWhile {
2     public static void main(String[] args) {
3         int i = 0;
4         while(i < 10) {
5             System.out.println("Hello World!");
6             i = i + 1;
7         }
8     }
9 }
10
```

The screenshot shows the BlueJ IDE with a Java class named `PrintHelloWorldWhile`. The code is as follows:

```
1 public class PrintHelloWorldWhile {
2     public static void main(String[] args) {
3         int i = 0;
4         while(i < 10) {
5             System.out.println("Hello World!");
6             i = i + 1;
7         }
8     }
9 }
10
```

Two breakpoints are set in the `main` method:

- Line 3: `int i = 0;`
- Line 5: `System.out.println("Hello World!");`

# Loops & Debugging (4)

We now modify the **PrintHelloWorldWhile** program we have just created into **PrintHelloWorld**:

```
public class PrintHelloWorld {  
    public static void main(String[] args) {  
        for(int i = 0; i < 10; i++)  
            System.out.println("Hello World!");  
    }  
}
```

Note: You will need to save this as a new class in BlueJ

Note: `i++` is an abbreviation for `i = i + 1`

## Loops & Debugging (5)

Again, before we run the code

```
for(int i = 0; i < 10; i++)  
    System.out.println("Hello World!");
```

we should be able to work out how many times "Hello World!" is printed to the terminal window by quickly drawing out a table.

counter i (at start of loop)	i < 10	Output to terminal	i after the increment (i++)
0	true	Hello World!	1
1	true	Hello World!	2
2	true	Hello World!	3
3	true	Hello World!	4
4	true	Hello World!	5
5	true	Hello World!	6
6	true	Hello World!	7
7	true	Hello World!	8
8	true	Hello World!	9
9	true	Hello World!	10
10	false		

## Exercise 1: InterestCalculator3

Using the **InterestCalculator2** class from last week, refactor the code to ask the user for the initial balance value and then use a `for` loop to calculate the interest (`current balance - initial balance`) after 10 years. Output a polite message informing the user of the current balance and interest earned each year.

## InterestCalculator4

Rewrite the code using a `while` loop to terminate when the current balance is double the initial balance. Again, notify the user of balance and interest where necessary.

Consider why a `for` loop was chosen for InterestCalculator3 and a `while` loop for InterestCalculator4.



## Marked Exercise 2: Powers

**Note:** *This class is the second marked exercise and you are required to complete it and show the working program, explaining step-by-step execution, by the 13<sup>th</sup> February.*

*Make sure you have backed up your work on a memory stick or similar.*

Write a program, class **Powers**, that prints all powers of a given  $n$  from  $n^1$  up to  $n^{15}$  without using class Math. Your program should follow the pattern given in the next slide and available from

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

Hints:

Unlike in Python, there is no `**` power operation in Java.

So, use a `for` loop and then multiply with a variable to store the "current" value inside the loop.

Remember to choose an appropriate integer datatype.

# Marked Exercise 2: Powers (2)

```
import java.util.Scanner;
public class Powers {
    public static void main(String[] args) {
        printPowers(2);
        printPowers(-1);
        printPowers(10);
        System.out.println("Enter a number: ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        printPowers(n);
    }

    public static void printPowers(int n) {
        System.out.println("Powers of " + n);

        // INSERT YOUR CODE HERE
    }
}
```

## Marked Exercise 3: BalancedParentheses

Implement the class **BalancedParentheses** with a method `isBalanced` that receives a `String` and checks whether the brackets in it are balanced.

Your program should follow the pattern given in the next slide and available from

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

**Note:** *The **BalancedParentheses** class is the third marked exercise and you are required to complete it and show the working program, explaining step-by-step execution, by the 13<sup>th</sup> February. Make sure you have backed up your work on a memory stick or similar.*

# Marked Exercise 3: BalancedParentheses (2)

```
import java.util.Scanner;
public class BalancedParentheses {
    public static void main(String[] args) {
        checkParentheses("(a + b) * t/2 * (1 - t)");
        checkParentheses("(a + b) * t)/(2 * (1 - t)");
        checkParentheses("a + ((a + b) * t)/(2 * (1 - t))");
        System.out.println("Enter an expression: ");
        Scanner scanner = new Scanner(System.in);
        String s = scanner.nextLine();
        checkParentheses(s);
    }
    public static void checkParentheses(String s) {
        System.out.println(s + " is "
            + (isBalanced(s) ? "" : "not ")
            + "parentheses balanced");
    }
    public static boolean isBalanced(String s) {
        // INSERT YOUR CODE HERE
    }
}
```

## Marked Exercise 3: Balanced Parentheses (3)

For example, consider the expression

$$((a + b) * t / 2 * (1 - t))$$

There are three ( and two ). The parentheses are *unbalanced*. This kind of typing error is very common with complicated expressions. Now consider the expression

$$(a + b) * t) / (2 * (1 - t))$$

This expression has three ( and three ), but it still is not correct. In the middle of the expression,

$$(a + b) * t) / (2 * (1 - t))$$

there is only one ( but two ), which is an error. In the middle of an expression, the count of ( must be greater than or equal to the count of ), and at the end of the expression the two counts must be the same.

# Balanced Parentheses: Hints

Here is a simple trick to check whether parentheses are balanced or not. Use a single counter when scanning the expression. Start with 1 at the first opening parenthesis, which is not necessarily the first character.

Then, add 1 whenever you see an opening parenthesis, and subtract 1 whenever you see a closing parenthesis.

If the count ever drops below zero, or is not zero at the end, the parentheses are unbalanced. For example, when scanning the previous expression, you would get the following:

$(a + b) * t) / (2 * (1 - t)$

1      0      -1

and would find the error.

# Home Work

## Java for Everyone by C. Horstmann

Read Chapters 4 & 5 (except 5.9), which are available online from

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

and complete the following exercises:

- Exercise R4.15
- Exercise R4.18
- Exercise R4.19
- Exercise P4.1
- Exercise P4.3
- Exercise P4.9