Chapter 2. Console Handling I: Basic Output

A significant part of many programs is the ability to present output to the user, via the interface with which the user is interacting with the program. This can be textual output to the console. In this chapter, exercises are presented with which the reader can practice and sharpen their skills in basic and formatted console output, using in particular the methods `System.out.printf`, `System.out.println` and `System.out.print`. Also, the basic use of string, integer and floating point variables appears in this chapter as the presented exercises pertain to outputting data of such types.

Output using the `System.out.printf` method

1. Predict the output of this program. Also explain the code.

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

Output: `Hello!`
Code Explanation: We used the `System.out.println` method to output the string "Hello" to the console. All we had to do was to pass the string, enclosed in double quotes to the `println` method.

2. Predict the output of this program. Also explain the code.

```
public class FormattingExercises{
  public static void main(String[] args){
    String var01 = "Hello!";
    System.out.printf("%s", var01);
  }
}
```

Output: `Hello!`
Code Explanation: We used the `System.out.printf` method to output the contents of a variable named `var01` of type `String` to the console.
Looking at the parameters passed to the `printf` method, we see that the first parameter is the double-quote bounded string `"%s"` and the second parameter is the name of a variable, in this case `var01`. The relationship between the 1st and subsequent parameters is as follows: The first parameter in a call to `printf` can contain any text that we want to be output, mixed in with "formatting strings", such as `%s`, which stand in for the values of `String` variable names that we pass along to `printf` as subsequent parameters to the 1st. In this exercise, we have one formatting string presented in the first parameter and we also pass the name of the variable that it stands in for to `printf`.
The number of formatting strings in the 1st parameter must always be matched by an equal number of variable names passed in as parameters 2…*n*.

3. Predict the output of this program. Also explain the code.

```
public class FormattingExercises{
  public static void main(String[] args){
    String var01 = "Hello!";
    String var02 = "How are you?";
    System.out.printf("My friend! %s %s", var01, var02);
  }
}
```

Output: `My friend! Hello! How are you?`
Code Explanation: Looking at the parameters passed to the `printf` method, we see that the first parameter is `"My friend! %s %s"` and the second and third parameters are the names of variables, in this case `var01` and `var02`. As stated in the solution to the preceding exercise, the first parameter passed to `printf` can contain any text that we want to be output and if desired we can mix in with that text "formatting strings", such as `%s`, which stand in for the values of variable names that we pass along to `printf` as subsequent parameters to the 1st parameter. In the 1st parameter we have the text *"My friend!"* and within the same parameter we have two formatting strings, `%s`, a space and again `%s`, which indicates to us that our output string will be the string literal

	"My friend!" followed by whatever variable is substituted for the first `%s` and then a space (as seen in parameter 1) and then the value of whatever variable is to be substituted for the second `%s`. It should be explicitly stated that the order of replacement of formatting strings in parameter 1 is the order in which parameters 2…*n* appear, i.e. the first formatting string is replaced by the value of parameter 2, the 2nd formatting string is replaced by parameter 3 the 3rd formatting string is replaced by parameter 4 and so on. The resultant output is as noted above.

4.	Predict the output of this program. Also explain the code.
	`public class FormattingExercises{`
	`  public static void main(String[] args){`
	`    String var01 = "Hello!";`
	`    System.out.printf("%s %s", var01, var01);`
	`  }`
	`}`

	Output: `Hello! Hello!` **Code Explanation:** The explanation of this code is essentially the same as that for the preceding exercise; only in this case we have chosen to have the same variable repeated for each of the two string formatting elements `%s` and `%s` in our 1st parameter.

5.	Predict the output of this program. Also explain the code.
	`public class FormattingExercises{`
	`  public static void main(String[] args){`
	`    String var01 = "Hello!";`
	`    String var02 = "How are you?";`
	`    System.out.printf("%s\n%s", var01, var02);`
	`  }`
	`}`

	Output: `Hello!` `How are you?` **Code Explanation:** We see that the output is on two lines. This is due to the effect of the formatting parameter **`\n`** which we put between the two formatting placeholders in the first parameter. `\n` essentially means "put a newline here". This is the effect that we have seen in our output.

6.	Predict the output of this program. Also explain the code.
	`public class FormattingExercises{`
	`  public static void main(String[] args){`
	`    String var01 = "Hello!";`
	`    String var02 = "How are you?";`
	`    System.out.printf("%s\t%s", var01, var02);`
	`  }`
	`}`

	Output: `Hello!  How are you?` **Code Explanation:** We see that the output has a tab between the two strings that have been output. This is due to the effect of the formatting parameter **`\t`** which we put between the two formatting placeholders in the exercise. `\t` means "put a tab here". This is the effect that we have seen in our output.

7.	I have two hardcoded `String` variables defined as follows:
	`String firstName = "John";`
	`String lastName  = "Doe";`
	Write a program which will use a `printf` statement to output these two `String` variables to the console on the same line with a single space in-between them.

	`public class FormattingExercises {`
	`  public static void main(String[] args) {`
	`    String firstName = "John";`
	`    String lastName  = "Doe";`
	`    System.out.printf("%s %s",firstName, lastName);`
	`  }`
	`}`

8.	Predict the output of this program. Also explain the code. `public class FormattingExercises{`
	Output: `2` **Code Explanation:** We are introduced here to a new formatting parameter, **`%d`**. The parameter **`%d`** is used for

	`public static void main(String[] args){`
	`  int var01=2;`
	`  System.out.printf("%d", var01);`
	`}`
	`}`
		integer values. In all other aspects, the description of the functioning of the `printf` method is the same as in the preceding exercises.

9.	Predict the output of this program. Also explain the code.
	`public class FormattingExercises{`
	`public static void main(String[] args){`
	`  double var01=2.55;`
	`  System.out.printf("%f", var01);`
	`}`
	`}`
		Output: `2.550000`
		Code Explanation: We see here the formatting parameter, `%f`. The parameter `%f` is used for floating point values. In all other aspects, the description of the functioning of the `printf` method is the same as in the preceding exercises. Observe that the output is written to 6 decimal places. This is the default output format of the `%f` formatting parameter. We see in a later chapter how to modify the precision of the output.

10.	Predict and explain the output of this program.
	`public class FormattingExercises{`
	`public static void main(String[] args){`
	`  double var01=2.123456789;`
	`  System.out.printf("%f", var01);`
	`}`
	`}`
		Output: `2.123457`
		Explanation: While we did indeed enter a number which has 9 decimal places, as previously stated, the `printf` method by default only prints values out to 6 decimal places, rounding the last digit if necessary. In a later chapter, we see how to modify the precision of the output.

Output using the `System.out.print & System.out.println` methods		
11.	Write programs which output to the console the text *"Hello World"* followed by a newline character using `System.out.print` and `System.out.println` respectively.	

	`public class FormattingExercises {`
	`public static void main(String[] args) {`
	`  System.out.println("Hello World");`
	`}`
	`}`
	-or-
	`public class FormattingExercises {`
	`public static void main(String[] args) {`
	`  System.out.print("Hello World\n");`
	`}`
	`}`	

	Note: You see that a `println` automatically adds a line terminator to the output. If you choose to use `print`, then you yourself have to add the newline character `'\n'` to the data being output in order to get the same output as `println`.	

12.	Write a program which uses `System.out.println` to output the text *"Hello World"*, inclusive of the double quotes, to the console.
	Hint: use the `\` *escape sequence.*
		`public class FormattingExercises {`
		`public static void main(String[] args) {`
		`  System.out.println("\"Hello World\"");`
		`}`
		`}`
		Note: See the *escaping* of each double quote using the `\` in each instance.

13.	Write a program which outputs the exact phrase "*Hello World*\n" inclusive of the "\n".	`public class FormattingExercises {`
		`public static void main(String[] args) {`
		`  System.out.println("Hello World\\n");`
		`}`
		`}`

Outputting from a variable using `System.out.println`		

14.	I have two hardcoded `String` variables defined as follows:
	`String firstName = "John";`
	`String lastName  = "Doe";`
	Write a program which will use a `println` statement to output these two `String` variables to the console on the same line with a single space
		`public class FormattingExercises {`
		`public static void main(String[] args) {`
		`  String firstName = "John";`
		`  String lastName  = "Doe";`
		`  System.out.println(firstName + " " + lastName);`
		`}`
		`}`

	in-between them.	

15.	I have two `String` variables defined as follows:
	`String firstName = "John";`
	`String lastName  = "Doe";`
	Concatenate these strings, with a space in-between them, putting the resulting concatenation into a single `String` variable named `fullName` and then output the concatenated string to the console.
		`public class FormattingExercises {`
		`  public static void main(String[] args) {`
		`  String firstName = "John";`
		`  String lastName = "Doe";`
		`  String fullName = firstName + " " + lastName;`
		`  System.out.println(fullName);`
		`  }`
		`}`

E1	The reader should study the class `DecimalFormat` to see how numerical formatting rules can be applied to the methods `System.out.println` and `System.out.println`.	