Ulster University

COM410 Programming in Practice

A2.3 Input, Output and Files

Text Output

- Output in Java is provided by the **`System.out`** object which provides methods **`println()`** and **`print()`**.

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

    System.out.println("Hello");
    System.out.println("Goodbye");

    System.out.print("Hello");
    System.out.print("Goodbye");
}
```

```
Hello
Goodbye
HelloGoodbye
```

- **`System.out.println()`** adds a newline character to the output
- **`System.out.print()`** keeps position on the same line for the next output

Formatted Text Output

- Formatted output in Java is provided by the `System.out.printf()` method
- Most common form takes a format string and a list of arguments

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

    String module = "COM410";
    int classSize = 180;
    double averageMark = 61.25;

    System.out.printf("Welcome to %s%n", module);
    System.out.printf("We have %d students and the average mark is %f%%", classSize, averageMark);
}
```

```
Welcome to COM410
We have 180 students and the average mark is 61.250000%
```

- `%s` embeds a string, `%d` embeds a decimal integer, `%f` embeds a floating-point number
- `%n` (or `\n`) inserts a newline character, `%%` (or `\%`) inserts a percentage character

Field Width

- Add a field width to specify the number of characters required

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

    double averageMark = 61.25;

    System.out.printf("The average mark is %f%% \n", averageMark);
    System.out.printf("Average mark to 2 decimal places is %.2f \n", averageMark);
    System.out.printf("In 6 characters with one decimal place is %6.1f \n", averageMark);
    System.out.printf("Padded with leading zeroes is %06.1f \n", averageMark);
}
```

```
The average mark is 61.250000%
Average mark to 2 decimal places is 61.25
In 6 characters with one decimal place is   61.3
Padded with leading zeroes is 0061.3
```

Field Width

- The field width can be used with any output type
- Output is right justified within the field by default, add the – flag for left justified

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

    System.out.print("*");
    System.out.printf("%50s", "Right justified to 50 character places");
    System.out.print("*\n*");
    System.out.printf("%-50s", "Left justified to 50 character places");
    System.out.println("*");
}
```

```
*            Right justified to 50 character places*
*Left justified to 50 character places              *
```

Keyboard Input

- Need to import the **`Scanner`** object and create a new instance of it
- Use the Scanner methods **`nextLine()`** or **`nextInt()`** to collect data from the user

```
What is your name? > Adrian
You entered Adrian
What is your age? > 21
You entered 21
```

```java
import java.util.Scanner;

public class Demo {

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

        System.out.print("What is your name? > ");
        String yourName = keyboard.nextLine();
        System.out.printf("You entered %s \n", yourName);

        System.out.print("What is your age? > ");
        int yourAge = keyboard.nextInt();
        System.out.printf("You entered %d \n", yourAge);
    }
}
```

Scenario

- Revisit the **TownChallenge** project and make modifications so that
    i. The names of the towns are provided by the user in response to an on-screen prompt
    ii. The results of the fixtures are presented on screen in a neatly presented table created using `System.out.printf()` statements and appropriate field widths.

Using Text Files for Input

- Need to import the `Scanner` and `File` objects and create new instances
- Note the path to the file and the use of the Exception handler (more later)

```
import java.util.Scanner;
import java.io.File;

public class Demo {

    public static void main(String[] args) throws Exception {

        File file = new File( pathname: "src/textfile.txt");
        Scanner fileInput = new Scanner(file);

        while (fileInput.hasNextLine()) {
            System.out.println("Reading... " + fileInput.nextLine());
        }
    }
}
```

textfile.txt

```
Line 1
Line 2
Line 3
Line 4
Line 5
```

Console output

```
Reading... Line 1
Reading... Line 2
Reading... Line 3
Reading... Line 4
Reading... Line 5
```

Using Text Files for Output

- Need to import the **`FileWriter`** object and create new instance

```
import java.io.FileWriter;

public class Demo {

    public static void main(String[] args) throws Exception {

        FileWriter file = new FileWriter( fileName: "src/outputfile.txt",  append: true);

        file.write( str: "Line 1 written to the file\n");
        file.write( str: "Line 2 written to the file\n");
        file.close();
    }
}
```

outputfile.txt

```
Line 1 written to the file
Line 2 written to the file
Line 1 written to the file
Line 2 written to the file
```

(Output shown is after running the code twice)

- Again, note the path to the file and the use of the Exception handler (more later)
- **`FileWriter`** constructor takes a boolean parameter to specify append mode

Challenge

In your **TownChallenge** project, modify the class `TownChallenge` to provide functionality as follows.

i. The application should read the eight town names from an input file called *towns.txt* and store them in the `towns` array

ii. After reading the collection of towns, the application should ask the user how many sets of results are required and accept this numeric value from the keyboard

iii. The program should generate as many sets of random results as were requested by the user. Note that all sets of results will consist of the same fixtures.

iv. All results (and home/draw/away summaries) should be written to a file called *results.txt* such that every time the program is run, new output is added to the end of the file. The information in the results file does not need to be formatted (i.e. as a table)