CS-115 Lab Class 3 — 10/02/2020

☐ Task 3.1

The goal of this task is to read data from a text file and process it. You will need to use formatted output for this task (this is new material). At the end of this lab sheet there are some hints about using Scanner and how to format data.

Download the following file by right clicking:

student_data.txt

Where should you put this file? Well that depends on how you are running your program (i.e., if you are using Eclipse or the command line). Java will read files (by default) from the *current working directory*. If you are using the command line then store this file in the same directory as your java program (i.e., where your source files and class files are stored for this task). If you are using Eclipse, then store (import) your text file into the *root directory* of your project. By default Eclipse sets this as the *current working directory* (do not place the data file in the *src* folder).

The data file contains (fictional) data about student marks for a university course. Each line contains a single student record where each field is separated by a single space character. The fields are:

- Student ID (an integer)
- First Name (a String without any spaces)
- Surname (a String without any spaces)
- Coursework 1 Mark (an integer between 0 and 100)
- Coursework 2 Mark (an integer between 0 and 100)
- Exam Mark (an integer between 0 and 100)

Your task is to write a Java program that reads this data and calculates the final marks.

Each of the mark fields are percentages expressed as integers (between 0 and 100). The final mark is calculate as follows:

- Coursework 1 is worth 10%.
- Coursework 2 is worth 10%.
- Exam is worth 80%.

For example: if Coursework 1 is 76, Coursework 2 is 88, and Exam is 60, then the final mark would be $(76 * 0.1) + (88 * 0.1) + (65 * 0.8) = 68.4$.

You need to read the data one line at a time, calculate the final mark for that student and output the final mark to **1 decimal place** along with the student details in the following format:

```
ID1, Surname1 Firstname1: Final_Mark1
ID2, Surname2 Firstname2: Final_Mark2
ID3, Surname3 Firstname3: Final_Mark3
...
```

The first few lines of your output should look like:

```
5001, Fowler Kevin: 55.3
5002, Phillips Carlos: 42.9
5003, Webb Dianne: 67.4
5004, Morales Justin: 24.6
...
```

Finally, you must output a nice message in the case that the file cannot be found. To do this you will need to catch the `FileNotFoundException` that the Scanner constructor can throw. Exit gracefully with an error code.

☐ Challenge Task 3.2

Format the output so that it is better aligned:

```
5001       Kevin      Fowler:        55.3
5002       Carlos     Phillips:      42.9
5003       Diane      Webb:          67.4
5004       Justin     Morales:       24.6
```

This can be done by using carefully selected format specifiers but you will need to read more about them (see the link at the end of this document).

☐ Challenge Task 3.3

Output the data in order of increasing final mark. For this you will need to create a Student class to store the data, keep all the student object in some form of array, sort the array and then print the data out in sorted order.

This is not an easy task, but equally it should not be impossible for you.

Scanner Hints

You can create Scanner objects in 3 different ways.

- Based on `System.in`:

  ```
  Scanner in = new Scanner(System.in);
  ```

  This will allow you to read and parse data from the standard input stream (i.e., the keyboard).

- Based on a text file:

  ```
  File f = new File("Some_File.txt");
  Scanner in = new Scanner(F);
  ```

This will allow you to read and parse data from the text file named `Some_File.txt`.

- Based on a string object:

```
String s = "Bob Alice Peter 42 54''
Scanner in = new Scanner(s);
```

This will allow you to read and parse data from the String `s`. This might be helpful as you could read an entire line with one Scanner then use a second Scanner to parse the individual fields in the line.

Formatted Numbers

You can turn floating-point numbers (i.e., doubles like 1.55000005) into a String version to 1 decimal place using the String class's format method:

```
double d = 1.55000005;
String s = String.format("%.1f", d);
```

The variable `s` will now be the String `"1.6"`. Here the first argument of `String.format` is string which contains format specifiers. A format specifier is a "hole" which is to be formatted in a specific way. The follow arguments are data which will be dropped into the holes and formatted.

Format specifiers start with a percentage sign. The format specifier `%.1f` means it is a hole for a floating-point number that will be formatted to 1 decimal place.

You can use format specifiers in a more advanced form. Here is an example

```
double d = 1.55000005;
int age = 37;
String name = "Bob";

String s = String.format("Hello %s, you are %03d and d is %.2f",
                name, age, d);
```

The variable `s` will now be the String `"Hello Bob, you are 037 and d is 1.55"`.

The format specifier `%s` is a hole for a String. The format specifier `%03d` is a hole for a decimal number (i.e., an integer) and will be formatted so that it is 3 characters long (and use leading zeros if needed). Finally, the format specifier `%.2f` is a hole for a floating-point number that will be formatted to 2 decimal places.

There is also a method `System.out.printf` that works in a similar way. Find out more at `https://docs.oracle.com/javase/tutorial/essential/io/formatting.html`