MARKED Exercise Sheet

CE152

MARKED LAB EXERCISE 2

Introduction

This sheet contains a ***marked*** exercise. The marked exercise requires you to **submit** the project as a zip file to FASER (CE152, Lab Exercise Week 21). You must also **demonstrate** your program to one of the GLAs or module supervisors. Please check that your mark is recorded correctly. You can demonstrate your solution during Week 21 or Week 22. This marked exercise will constitute 8% of your total grade.

This exercise will focus on **Unit 4 (Arrays), 5 (Files and exceptions) and 6 (2D Graphics)**. Please watch the corresponding videos on Moodle in preparation.

You are expected to be aware of the content of these Units during your demonstration. *If you are unable to explain your code this will affect your mark*.

Please remember that submitting someone else's work or work generated by an AI as your own may be **considered an academic offence**. Refer to the appropriate guidelines or ask if you are uncertain about what this means.

Create today's project

Start IntelliJ and create a project named markedLab2.

MARKED EXERCISE

This sheet requires you to complete Exercise 1 [20%], Exercise 2 [30%], Exercise 3 [40%], comment your code [5%] and export it as an archive [5%].

Exercise 1 [Total 20%]

Exercise 1A [5%]

Create a class called `Letter2D`.

Add three `private` variables to the class. An `int` called `x`, an `int` called `y`, and a `char` called `c`.

Exercise 1B [5%]

Add a constructor that receives values for `x`, `y` and `c` and sets the variables accordingly.

Exercise 1C [10%]

Add a `public` get method for each of the three variables: `getX` should return `x`, `getY` should return `y` and `getChar` should return `c`.

Furthermore override `toString()` that returns a `String` consisting of the character `c` and the coordinates `x` and `y` all separated by commas.

Exercise 2 [Total 30%]

Exercise 2A [5%]

Create a class called `Letter2DI0`.

Add two `static` variables that are `private` and `final`, of type `int` with value 500. These variables will store the maximum values the x and y coordinates of the `Letter2D` objects you create may have. Name them in accordance with Java naming conventions.

Exercise 2B [25%]

Add a `public` and `static` method called `writeRandomLetters` that returns `void`. The method should have a `String` and `int` parameter. The `String` specifies the name of the file to write to. The `int` the number of `Letter2D` Objects to create and write. Now add code to the body of the method that writes the specified number (the `int` method parameter) of `Letter2D` Objects to the specified file (the `String` method parameter). Generate the coordinates randomly between zero and the maxima defined by the variables you created in 2A and also generate a random character between a and z. Write the objects to the file using their `toString()` method. If you did not complete Exercise 1 write two `int` in the specified range per line separated by commas.

Additionally add a `public` and `static` method called `readLetters` that returns `Letter2D[]`. The method should have a `String` and `int` parameter. The `String` specifies the name of the file to read from. The `int` the number of lines to read from the file (each line contains the data of one `Letter2D` Object). Now add code to the body of the method that converts each line in the file to a `Letter2D` Object with the specified coordinates. Store them in a variable of type `Letter2D[]`. Make sure you do not read more than the number of elements specified. Return the elements that you read as a variable of type `Letter2D[]`. If you did not complete Exercise 1, read the `int` values into a 2D array (`int[][]`).

Exercise 3 [Total 40%]

Exercise 3A [5%]

Create a class called `Letter2DDisplay`. Extend this class from `JComponent`.

Add a `private` variable of type `Letter2D[]`. If you have not completed Exercise 1 replace `Letter2D[]` with a 2D `int` array in all Exercises.

Exercise 3B [5%]

Add a constructor to the class that receives a `Letter2D[]` parameter. Initialise the variable you created in 3A with this parameter.

Exercise 3C [15%]

Override the `paintComponent` method. In the method iterate over the `Letter2D[]` variable and draw the corresponding letter for each element at the coordinates stored in the `Letter2D`. If you are using the `int[][]` choose a colour and radius randomly and draw a corresponding oval.

Exercise 3D [15%]

Create another class called `MyMain` with a main method. In this main method first write 100 `Letter2D` (or `int[][]`) to a file, then read them again into a corresponding array.

Create a `JFrame` with size 500 x 500 pixels and add a `Letter2DDisplay` object to it. Provide the array you just read to the `Letter2DDisplay` constructor. Make the `JFrame` visible.

You can use the code for creating the `JFrame` and writing and reading using `Letter2DIO` below:

```
Letter2DIO.writeRandomLetters("test", 100);
JFrame f = new JFrame("Letter2D Display");
f.setSize(500, 500);
f.add(new Letter2DDisplay(Letter2DIO.readLetters("test", 100)));
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
```

You will have to surround the code with an appropriate `try/catch` block and place it in your `main()` method.

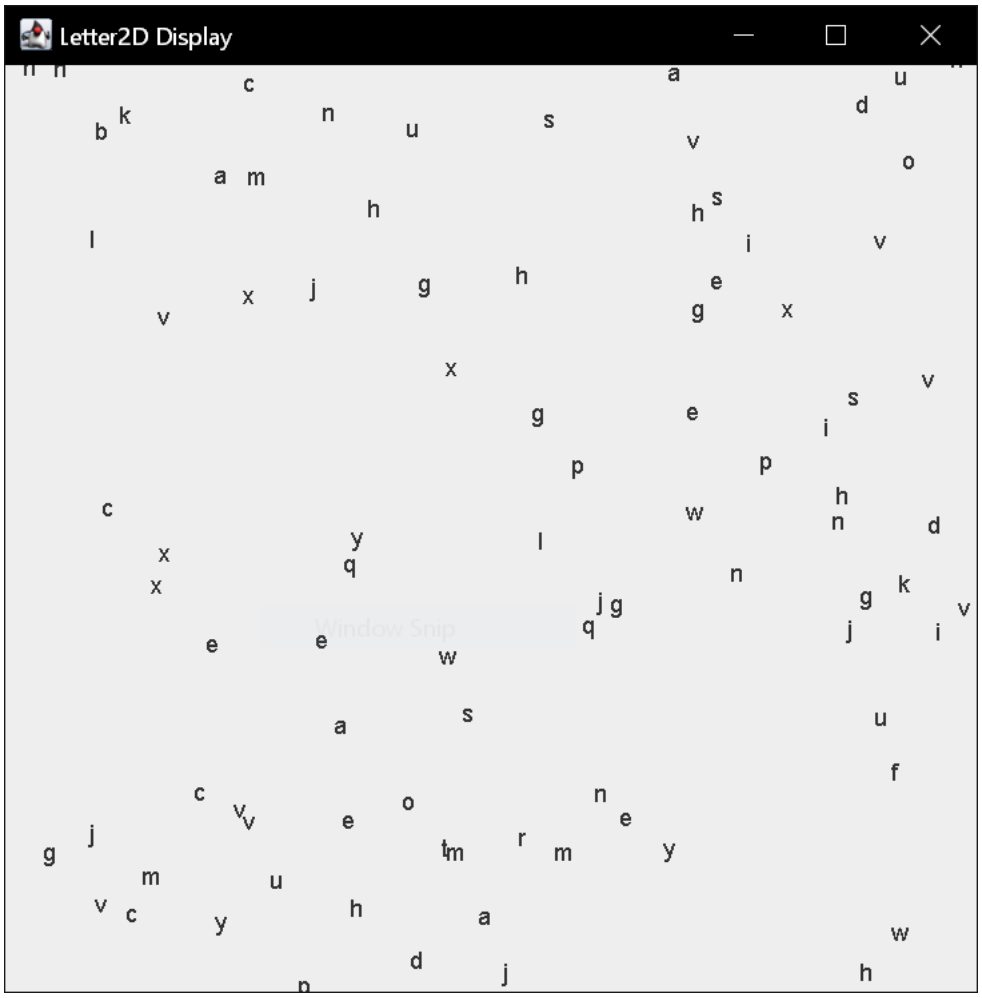

Figure 1: An example JFrame (500x500) with 100 Letter2D in black colour.

Comments [5%]

In this exercise you should practise commenting your code. To receive the marks please provide **one comment per method** explaining the **purpose** of the method and your **approach** to providing the requested functionality.

Exporting and submitting [5%]

Please read this entire section before submitting your code to Faser.

Using IntelliJ, in the menu File, Export choose Project to zip file. . . . Note the folder you exported to and submit this zip file to Faser. **Please check that you have uploaded the correct file**: download it from Faser, unzip it and check if the source code you wrote is contained in the archive.

Some versions of IntelliJ do not have the export to zip file item on the menu. In this case, you have two options:

1. File, Settings. . . , go to the Plugins tab, Installed pane, and tick the Android plugin. When you hit OK, IntelliJ will restart and the menu item should be there.
2. Just use any zip software to create a zip file of the project folder. Include the whole project – this is one level above `src` in a project that's set up in the standard way.

If you are not using IntelliJ you must still upload your project to Faser as a zip archive. Unfortunately, I cannot provide you with instructions on how to do this, but you can always use any zip software, as above.