CE203 ASSIGNMENT 1 2020

Credit:
30% of total module mark

Module Learning Outcomes Examined:
1) Demonstrate knowledge of core Java application libraries. 3) Write Java programs with interactive graphical user interfaces (GUIs). 5) Write Java programs that make efficient use of the Java collections package.

Submission Deadline:
Wednesday 25/11/2020 before 11:59:59 (Week 8)

Submission will be via the Faser online submission system

You should refer to the Undergraduate Students' Handbook for details of the departmental policy regarding late submission and university regulations regarding plagiarism; **the work handed in must be entirely your own**. It is expected that marking of the assignment will be completed within 3/4 working weeks of the submission deadline.

Introduction
This assignment involves writing a Java application using Swing and AWT GUI components and a Java collections data structure. This must be hand-coded by you. If you use IntelliJ Idea or any other IDE you must not use the GUI Designer or any similar tool. Additionally, you should not use any other external libraries not part of the Java SE API. The submitted program **must** be runnable without dependencies on any external libraries not part of the Java SE used for the module. The files should contain ***brief*** comments.

WARNING AND ADVICE ABOUT POSSIBLE ACADEMIC OFFENCES
Your solutions should be your own unaided work. You can make use of any of the programs from the CE203 lecture notes and the lab templates on Moodle. You may use any features from the Java Standard Edition Application Programming Interface (Java SE API) including those not covered in CE203.
You must **NOT** use any third-party classes (e.g. classes that are **not** provided as part of the Java SE used for this module). For more information, please see the University pages on plagiarism and the Academic Offences Procedures.
DO NOT COPY PROGRAM CODE FOR THIS ASSIGNMENT FROM ANOTHER STUDENT OR FROM THE INTERNET OR FROM ANY OTHER SOURCES. DO NOT LET OTHER STUDENTS COPY YOUR WORK

Part 1

Write a frame-based application that allows the user to store a list of inventory numerical IDs. A legal ID should comprise of six digits such as "987217"

The layout should be such that any buttons will be displayed at the bottom of the panel, user input is provided in a text field at the top and any system output will be displayed in the centre either directly on the panel or within a text area component. Code is provided that defines an incomplete ID class that implements the comparable interface. This class should be used/modified to store and refer to a specific Id. The application should provide features and corresponding buttons that allow the user to:

a) Add an ID to the list (the ID to be added would be provided in the text field at the bottom of the application).
b) Display all the IDs from the list.
c) Remove from the list all matching occurrences of an ID entered in the text field.
d) Sort the list of IDs in ascending order and display the contents of the list. Here please explore the use of Collections.sort() to automatically sort the contents of the list. See Java API and an example of using it here: https://www.geeksforgeeks.org/collections-sort-java-examples/ (note the use of generics).
e) Clear the list.

The list may contain duplicate entries. Ids may contain only whole digits, so should not contain any negative digits, or any other characters. Your IDs should be a fixed length of 6 digits.

For each action specified above an appropriate message should be displayed on the centre panel confirming the action, e.g.

"ID '126658' has been added to the list".

An appropriate error message should be displayed for any failed action e.g.

"The ID '12786n' was not added to the list as it is not a valid ID."

The list for storing the IDs should be implemented using an `ArrayList`

Part 2

A colour may be specified by its *RGB value* – the amount of red, green and blue it contains. These amounts are integers between 0 and 255. For example, red is represented by (255, 0, 0), blue by (0, 0, 255) and yellow (which is made up from red and green) by (255, 255, 0). White is (255, 255, 255) and black is (0, 0, 0). The `Color` class has a constructor with 3 arguments that allows a user to create a `Color` object by providing the RGB values.

Extend the application you developed for Part 1 so that it allows the user to specify RGB values in three additional text fields. These should be located next to the existing text field already defined for entering IDs into the application.

f) If, the application's button (corresponding to displaying the IDs in the list, defined in Part 1) is pressed, the IDs should be displayed in the colour specified by the RGB values entered in the new text fields.
g) If the RGB values entered are not valid integers or have values less than 0 or greater than 255, then the displayed text should be set to Black and an appropriate error message should also be displayed.

User input exceptions for both Parts 1 and 2 should be captured by the program using exception handling or acceptable error checking.

You should aim to make your user interface easy to use.

You many have defined more than one Java class for your application. It is normally good practice to save these in their own physical .java files. However, for this assignment all your defined classes should be contained in a single physical file. This file should contain a single public class named `CE203_<SID>_Ass1` containing your `main()` method from where you the application executes. You should name the file `CE203_<SID>_Ass1.java` corresponding to this name where **<*SID*>** should be replaced with your SID number. Please check that the code runs prior to submitting it.

Commenting the Programs

The programs should contain brief comments indicating precisely what each method in the code does and what each instance variable is used for. You should not write any comments stating what individual lines of code do. In places it may be appropriate to simply state what a block of code does (e.g. "check that the input is a valid ID").

The programs should be laid out neatly with clear indentation.

Program Testing Output

The program produced should be fully tested to demonstrate that each of the features you are asked to implement is working correctly.

You are therefore asked to include either a Word or PDF document where you clearly show for each feature the correct output of your code. This can be shown as a series of screenshots that demonstrate the correct response of the program to each button press or user input / button press event. For example, in Part 1 where you are asked to display all the IDs from the list, you should display a screenshot of the GUI where this is shown to be correctly displayed following the button press event. Equally in Part 2 where the user must enter valid numbers for the R, G, B values in each text field, a screenshot of the output of the program showing correctly, the list of IDs displayed in the colour specified is required.

Please briefly explain or label the screenshots for each feature being tested for example: 'b), c)…'

The Word of PDF document should include your registration number and should be named according to the following naming convention:
Assignment 1_Testing_RegNo_<*registration number*>.

Submission

You are required to submit the source code file `CE203_<SID>_Ass1.java` (where **<*SID*>** should be replaced with your SID number) and all the `.class` files generated by the compiler. You need to also include the Word/PDF file containing your program testing output. All the files should be placed in a single folder, which should then be *zipped* for submission to the online system, i.e. the submission should be a single file in *zip format*. **Other formats (e.g. *rar* format) and submissions without the source code will <u>not</u> be accepted and will result in a mark of zero.**

Assessors will need to be able to run your code as is without any modifications required. **So, you must make sure that your code is able to run correctly prior to submission.**

Marks will be awarded for following *all* the requirements (including naming of files, layout of the interface etc.) and applying object-oriented programming principles appropriately.

Marking Criteria

1. Program compiles and runs	**/20%**
2. Correct layout, display of GUI components and program outputs	**/20%**
- Correct display of output text in colour as specified in RGB text boxes	
3. Correct button functionality	**/20%**
- Adding IDs as specified	
- Display all IDs in list	
- Removal of IDs as specified	
- Sort IDs in ascending order as specified using Collections.sort()	
- Clear list as specified	
- Correct use of ArrayList java collections data structure	
4. Exception handling or other acceptable error checking approach	**/20%**
5. Overall elegance and correctness of code	**/20%**
- Commenting of code	
- Object-orientated approach including the use of Generics	
Total	**/100%**

For each of the 5 marking criteria, marks will be awarded out of 20% based on the following bands:

Poor: <=8%
Satisfactory: 9-10%
Good: 11-13%
Excellent: 14-16%
Outstanding: 17-20%