1. Object-Oriented Programming 2 – Graded Exercises, Series 1 (DOJ)

Exercise 1 – GUI

Implement the GUI below that has the following functionality:

- The “Big Java” text must always be written according to the selected font type (Serif, SansSerif or Monospaced), style (normal, italic or bold) and size.
  - The font type can be selected by using the combo box or the font face menu.
  - The font style can be selected by using the check boxes or the font style menu.
  - The font size can be selected by using the radio buttons or the slider.
- In the middle of the GUI, the size label always displays the current font size.
- All inputs must always remain **synchronized**. For example:
  - If the style “italic” is selected in the menus, the corresponding check box must programmatically be selected, and conversely;
  - If the size "large" is selected with a button, the slider must be set to the corresponding value, and conversely; etc.

Where appropriate, try to use binding operations. Hint.: If the window is too small to display the text "Big Java", you can adapt its size by calling `stage.sizeToScene`.

In Moodle, you will find the image ("`BigJava.jpg`") and a simplified basic structure for your program ("`FontViewer.java`") that you can use if you want:

`https://moodle.bfh.ch/mod/folder/view.php?id=487851`

Exercise 2 – MVC

Extend the counter example that we saw during the lessons on the MVC pattern as follows:

- Instead of a single counter, create stopwatches (at least two of them) with a HH:MM:SS display that can be started, stopped and reset (to zero). Each stopwatch must be contained in a separate window.
- To control them, define at least two control units with three buttons (start, stop, reset). Each control unit must control all the stopwatches (i.e. when the start button of one control unit is clicked, all watches start; when one of the stop buttons is clicked, all watches stop, and so on). Build one of your control units together with the stopwatch display (i.e. in the same window) and one in a separate window (therefore, you should have at least three windows in your application). Make various layouts.
- When the watches have been started, the start buttons must be disabled until the watches are stopped. Similarly, when the watches have been stopped, the stop buttons must be disabled until they are restarted. In one of the control unit, display a corresponding message "Watches are running", bzw. "Watches are not running".
- Where appropriate, use binding operations.

Use object serialization to control the behavior of the application by terminating and restarting:

- When any of the windows is closed, the application must terminate.
- When the application restarts, the stopwatches must not be reset to zero, but conserve the value they had when the application terminated.
- Similarly, the state of the GUI should not change when the application terminates and restarts:
  - Buttons that have been disabled by termination should still be disabled by restart.
  - If the stopwatches were incrementing by termination, they must be further incrementing, as soon as the application restarts, without any action needed on the start buttons.

Exercise 3 – Processing Data Files

For this exercise, use the given files `worldpop.dat` and `worldarea.dat`.

The file `worldpop.dat` contains a list of countries with their population. The file has been generated from a list of `CountryPopulation` objects, which contained the name of a country and the associated population value. The generation process was carried out by the following algorithm:

```
List<CountryPopulation> countries = ...;
try (DataOutputStream dos =
       new DataOutputStream(new FileOutputStream("worldpop.dat"))) {
  for (CountryPopulation country : countries) {
    dos.writeChars(country.getName() + "/");
    dos.writeInt(country.getValue());
  }
}
```

The file `worldarea.dat` contains a list of countries with their area (in km^2) and has been generated from a list of `CountryArea` objects in a similar way as `worldpop.dat`. Both files contain the same country names in the same order.

Implement an application with the following functionalities:

- At start, the application must read both files (use a `FileChooser` to select them) and convert them in two corresponding text files `worldpop.dat.txt` and `worldarea.dat.txt`, in which each line contains the name of a country and its associated value, separated with some whitespace characters. For example:

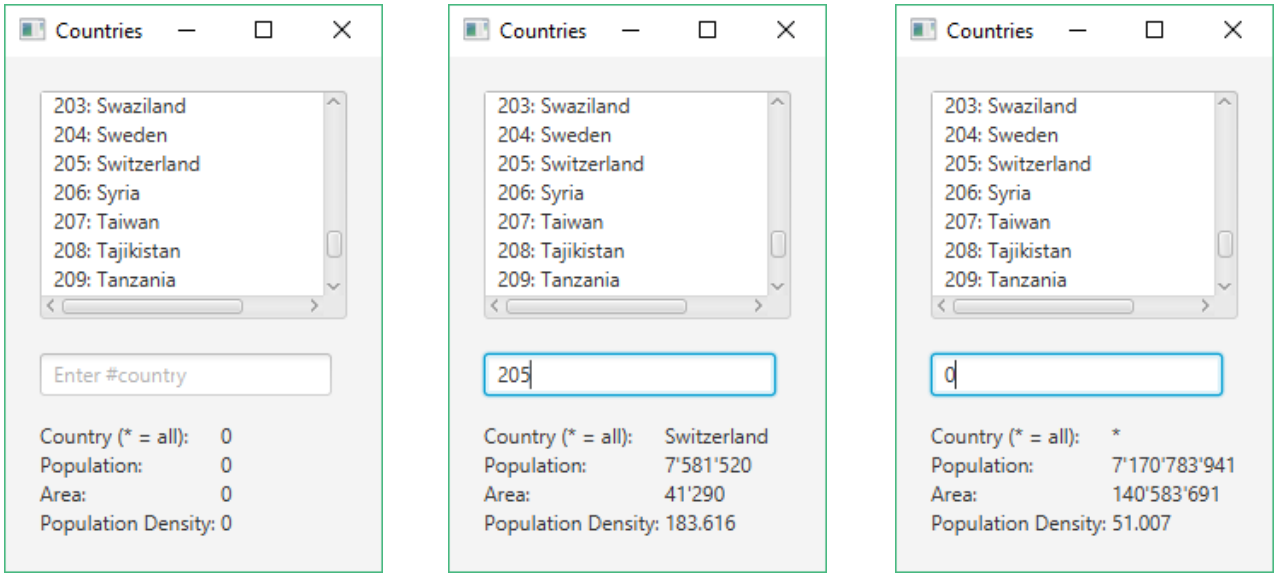
worldpop.dat.txt

```
Afghanistan 32738376
Akrotiri    15700
Albania     3619778
...
```

worldarea.dat.txt

```
Afghanistan 647500
Akrotiri    123
Albania     28748
...
```

- Then, the application must read both text files, compute the total population and the total area of all countries, and displays the corresponding results in the GUI shown below.
- In the GUI:
  - The text area contains the (numerated) name of the countries
  - The text field allows to specify the number of a country, whose name, population, area and population density will be displayed in the corresponding labels. If the value 0 is entered, the total population, area and density should be displayed.



Implement a correct and meaningful exception handling. If a file content is erroneous (i.e. a country or a value is missing), ignore that country.

Note: If you had some difficulties to convert the given binary files into text files, you could still test the rest of your program by using the files *worldpopHelp.txt* and *worldareaHelp.txt*, which are also given.

.... Other exercises are coming soon …