Exam F20 June 02, 2020

This exam contains 1 programming task, which weighs 60% of the final grade in only the VOP course. We have provided a *ZIP file* called *VOPExam20Exercise.zip*. *UNZIP* this file and open the maven project to see the packages containing the code snippets of the programming tasks. To open the project, go to *File -> open* in IntelliJ and navigate to the "pom.xml" file in the unzipped project folder.
Hint: *Take some time to read through the set of tasks before you start working on the solutions.*

Submission

At the end of the exam, the solution must be handed over to the Digital Exam:

- In IntelliJ go to *"File -> Export to Zip File…"*
- Make sure the file is named correctly. For example. "abcd19_VOP20_Exam.zip "
- Upload the zip file to Digital Exam. Remember to click Submit!

Task 1: 60%

In this task, we are going to implement a simple and a GUI interface for viewing sprinters of a marathon according to their countries. We will also implement features for also sorting the sprinters according to their attributes. We have provided a *comma-separated file* called *challenge.csv*. This file contains the result of sprinters at the 2016 Hong Kong Marathon. The file is taken from https://www.kaggle.com/melvincheung/hong-kong-marathon-2016?select=challenge.csv.

The file is "," comma-separated and each line represents information about a particular sprinter. Each line in the file has 11 fields as follows:

```
 1.  Overall Position,
 2.  Gender Position,
 3.  Category Position,
 4.  Category,
 5.  Race No,
 6.  Country,
 7.  Official Time,
 8.  Net Time,
 9.  10km Time,
 10. Half Way Time,
 11. 30km Time
```

These fields can be seen as the first entry in the *challenge.csv* file. The fields highlighted in red will be used in this task. Take care to note the positions of the fields.

Task 1a: Sprinter implements Comparable <Sprinter> 10 %

In this subtask, we have provided a class file called `Sprinter.java` in the package *sprinter*. An instance of the class should represent one entry in the *challenge.csv* file. Complete the implementation of the class so that it will contain:

- 6 Private variables for `raceNo(int), overallPosition(int), genderPosition(int), country(String), officialTime(String), halfTime(String).`

- Constructor which takes the 6 parameters corresponding to the 6 private variables and assign the parameters to the declared private variables.
- 6 `Getter` methods for all the 6 private variables. A `Getter` method should return the value of the private variable. An example of a `Getter` method is given below for `country(String)`:

```
public String getCountry() {
    return country;
}
```

- A `toString()` method. This method is already implemented but commented out. Uncomment this code and ensure that the called `Getter` methods in the `toString()` method corresponds to the ones you have created.
- A `compareTo()` method. This should be implemented so that the `raceNo` of two `Sprinters` objects are compared. Use the corresponding `Getter` methods for this implementation.
- A `main()` method for testing. This method is already implemented but the lines of code in the method are commented out. Uncomment the code under Task 1a and before Task 1b to test your implementation so far.

Example of correct output (They represent the first three entries in the *challenge.csv* file)

```
[2              Ethiopia        2:12:20         1:04:49         3
, 14            Kenya           2:12:14         1:04:48         2
, 21080         Kenya           2:12:12         1:04:48         1
]
```

Task 1b: Sorting with Comparator 5%

- Create a class with the signature `public class ComparatorHalfTime implements Comparator<Sprinter>` in the *sprinter* package.
- Implement the `compare()` method to compare two `Sprinter` objects by their `halfTime` values and if two objects have the same `halfTime`, they should be compared by their `raceNo` values.
- Create another class with the signature `public class ComparatorPosition implements Comparator<Sprinter>` in the *sprinter* package.
- Implement the `compare()` method to compare two `sprinter` objects by their `overallPosition` values and it the two objects have the same `overallPosition`, they should be compared by their `raceNo` values.
- Make sure to use the corresponding `Getter` methods in the `Sprinter` object coded in the solution file.
- In the `main()` method of the Sprinter class, uncomment the code under Task 1b to test your implementation so far.

Example of correct output:

```
[14             Kenya           2:12:14         1:04:48         2
, 21080         Kenya           2:12:12         1:04:48         1
, 2             Ethiopia        2:12:20         1:04:49         3
]

[21080          Kenya           2:12:12         1:04:48         1
, 14            Kenya           2:12:14         1:04:48         2
, 2             Ethiopia        2:12:20         1:04:49         3
```

]

Task 1c: ReadCSV implements Runnable 15%

For this task, we have provided a class file name `ReadCSV.java` in the *sprinter* package and an interface called `CallBackInterface` in the *callback* package. . This interface has two abstract methods `updateStatus(String message)` and `updateView()`. The `ReadCSV.java` class implements the `Runnable` interface and its methods are partially implemented. We have provided the following:

- Declared a private variable `callBack` of the type `CallBackInterface.`
- Declared a private variable `file` of the type `File.`
- Declared a private variable `map` of the type `map<String, Set>.`
- Implemented a constructor `ReadCSV(String fileName)` that instantiates the declared `callBack`, `map`, and `file` variables.
- Implemented an overloaded constructor with the signature `public ReadCSV(CallBackInterface callBack, File file).`
- Implemented the `run()` method that calls a `readFile()` method and some methods of the `callback` interface.
- A `Getter` method called `getMap()` that returns the instantiated `map` variable.

Complete the `ReadCSV.java` class Implementation as follows:

- Implement the `readFile()` method in the class to use the `java.util.Scanner` methods to read each line in the file *challenge.csv*. Remember that this file is comma separated "," and relevant fields to be read are provided in the introduction to this task.
  - Ensure that you enclose the input stream in a `try - catch` clause and close your input stream after use.
  - Make sure to catch the relevant exceptions that can be thrown.
  - Read each line in the *challenge.csv* file and parse the relevant fields of the line into their appropriate types to create a new `Sprinter` object.
  - Use the relevant `Getter` method of the `Sprinter` object to get the `Country` parameter and use this parameter to check if the Sprinter's `Country` exists in the `map` object. See an example for inspiration below:

    ```
    this.getMap().containsKey(sprinter.getCountry())
    ```

  - If the `Country` already exists, simply add the newly created `Sprinter` object to the sorted `set` of `Sprinter`s belonging to that particular `Country`.
  - If the `Country` does not exist in the `map`, create a new sorted `Set`, add the new `Sprinter` object to the new `Set` and put the `Country` and its corresponding `Set` object in the `map`.
- Implement the `run()` method that calls a `readFile()` method and some methods of the `callback` interface.
  - Before the `readFile()` is called, call the `updateStatus()` method of the `callBack` interface with a String value "`Reading Started`" and a newline character("`\n`").
  - After the `readFile()` method is called, call the `updateStatus()` method of the `callBack` interface to print "`Reading Completed`" with a newline character("`\n`").
  - Lastly, call the `updateView()` method of the `callBack` interface.
- To test your implementations, by executing the `main()` method of the `ReadCSV.java` class.

Example of correct output showing the beginning and ending the of execution (separated by “---”):

```
Reading Started


Reading Completed
---

, 30018     United States           3:57:17           0:54:39           1589
, 33002     United States           4:00:48           1:01:55           1756
, 33262     United States           3:06:03           0:42:52           161
, 33380     United States           3:14:28           0:45:43           291
, 36002     United States           4:42:41           1:01:27           3374
, 36026     United States           5:47:51           1:16:19           5364
, 36056     United States           4:58:30           1:03:03           3977
, 36447     United States           4:46:00           1:09:08           3524
]}
```

Task 1d: PrimaryController implements CallBackInterface, Initializable 30%

In this task, we have provided three class files and an *FXML* document namely *App.java*, *Main.java*, *PrimaryController.java* and *primary.fxml* respectively. The three class files can be found in the *app* package while the *primary.fxml* can be found in the *app* folder under the *resources* folder. The *PrimaryController.java* class is the controller class for the *primary.fxml* document. Both files will be used to implement this task.

Implement the `PrimaryController.java` class to implement the `CallBackInterface` and the `Initializable` interface as follows:

```
public class PrimaryController implements CallBackInterface, Initializable
```

- Declare a private variable of the type `File` called `selectedFile` as follows:

```
private File selectedFile;
```

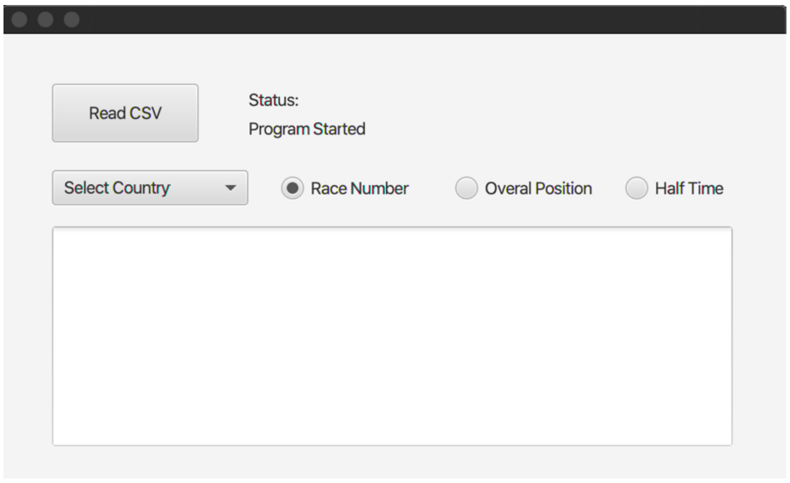
- Declare a private variable of the ReadCSV called readCSV as follows:

```
private ReadCSV readCSV;
```

- We have implemented the `readFile() actionHandler` and declared the `sortAction() actionHandler`:

Create the layout that can be seen below in *primary.fxml* document with the following items:

- A Button called "Read CSV"and associate it with a `readFile()` `actionHandler`.
- Two `Label`s. The first `Label` is for showing the title of the status message. Set the text of this label to *"Status:"* as shown in the figure. The second `Label` is for showing updates using the `updateStatus(String message)` method from the `CallBackInterface`. Set the text of this label to *"Program Started"* as shown in the figure.



- A `ComboBox` for showing the Countries in the sorted `Map` in the `ReadCSV.java` class. Set the prompt text of the `ComboBox` to *"Select Country".* Associate the `ComboBox` with the `sortAction()` `actionHandler.`
- `Three RadioButton`s for calling the relevant `Comparator<Sprinter>` for sorting the `Sprinter`s in a selected Country in the `ComboBox`. Make sure the `RadioButton`s are in the same toggle group. Associate the three `RadioButton` with the `sortAction() actionHandler.`
- A `TextArea` for showing the `Sprinter`s of a selected country in the `ComboBox.`

Implement the methods in the PrimaryController.java class as follows:

- In the `initialize()` method, set the `ComboBox`, `TextArea`, and the three `RadioButton`s to disable. An example for inspiration is given below:

```
cmbCountry.setDisable(true);
```

- Implement the `updateView()` so that it updates the items in the `ComboBox` with the `keySet` of the `map`. Make sure to use `PlatForm.runLater` for any update on the GUI. Use the following code as inspiration for setting the `ComboBox` with the `keySet` from the `map`:

```
cmbCountry.setItems(FXCollections.observableArrayList(readCSV.getMap().keySet())
);
```

- Implement the `updateStatus(String message)` so that the value of `message` is used to update the label whose text is set to *"Program Started"*. Make sure to use `PlatForm.runLater` for any update on the GUI. Check if the value of `message` is "`Read Completed`" (without " ") to enable the `ComboBox`, `TextArea`, and the three `RadioButton`s. An example for inspiration is given below for enabling the components:

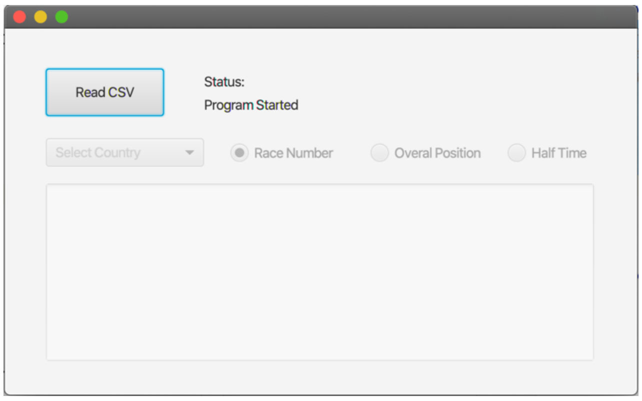
```
cmbCountry.setDisable(false);
```

- Implement the `sortAction() actionHandler` method so that it displays **only** the `Sprinters` from the country that is selected from the `ComboBox` to the `TextArea`. The `Sprinters` displayed in the `TextArea` should be **sorted** according to the selected option in the three `RadioButton`s and ensure that all your implementations for updating the GUI are wrapped using `Platform.runLater`. Implement this method as follows:
  - If the `RadioButton` for "Race Number" is selected (for example `rdRaceNumber.isSelected()`), the `Sprinters` should be sorted according to their `raceNo` values. (**N.B.** this is already implemented in the `Comparable` for the `Sprinter` Object).

    - If the `RadioButton` for ”Overal Position” is selected, the `Sprinter`s should be sorted according to their `overallPosition` values. (**N.B.** you will need to use the `ComparatorPosition.java` class to implement this function).
    - If the `RadioButton` for ”Half Time” is selected, the `Sprinter`s should be sorted according to their `halfTime` values. (**N.B.** you will need to use the `ComparatorHalfTime.java` class to implement this function).
- In the `readFile()actionHandler` method, uncomment the provided lines of code.
- Test your implementation by running the `Main.java` class in the *app* package. If you click on the Read CSV button, you should be prompted with a *File Explorer*. Navigate to the *challenge.csv* file and select it. Select a country from the `ComboBox` (for example *Australia*).

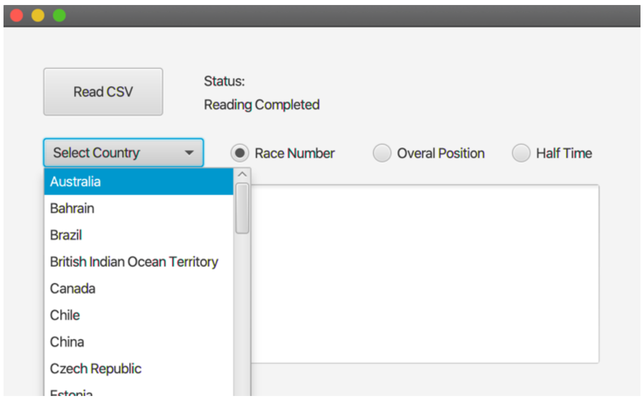
Example of correct outputs in Figure (a) – (c). Figure (c) shows the result for selecting *Australia* in the `ComboBox`:

(a)



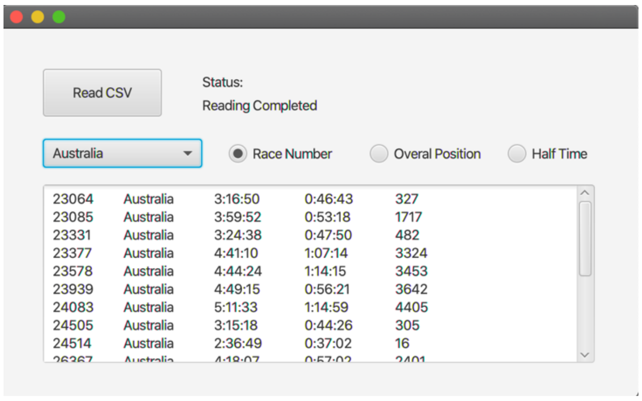
Status:
Program Started

(b)



Status:
Reading Completed

(c)



Status:
Reading Completed

```
23064   Australia   3:16:50   0:46:43   327
23085   Australia   3:59:52   0:53:18   1717
23331   Australia   3:24:38   0:47:50   482
23377   Australia   4:41:10   1:07:14   3324
23578   Australia   4:44:24   1:14:15   3453
23939   Australia   4:49:15   0:56:21   3642
24083   Australia   5:11:33   1:14:59   4405
24505   Australia   3:15:18   0:44:26   305
24514   Australia   2:36:49   0:37:02   16
26367   Australia   4:18:07   0:57:02   2401
```