Lab Exercise 4 – Event Handling

Computer Science 2334

Due by: Friday, 10 March 2017, 11:59 pm

This lab is an individual exercise. Students must complete this assignment on their own.

Objectives:

1. To learn how to use event handling in Java by completing an information display program.
2. To learn how to modify a class such that it implements and uses **ActionListener** as the event listener on Graphical User Interface (GUI) components.
3. To learn how to implement `actionPerformed()` to handle multiple events from components.
4. To see how to use a **LinkedHashMap**.
5. To demonstrate this knowledge by completing a series of exercises.

Instructions:

This lab exercise requires a laptop with an Internet connection. Once you have completed the exercises in this document, you will submit it for grading.

Make sure you read this lab description and look at all of the source code posted on the class website for this lab exercise before you begin working.

Assignment:

Event handling is an important feature of many programming languages, including Java. Event handling is used extensively in GUI-based programs to make them responsive to user gestures (such as clicks on the GUI's buttons) and to keep the graphical output of these programs in sync with data updates behind the scenes. Moreover, event handling will be used in some of your projects for this course and covered on the second and third exams. Carefully inspect how it works and the documentation comments included in the code.

1. Download the `Lab4-eclipse.zip` project archive from the class website. Import the project into your Eclipse workspace using the slides from Lab 2. You will submit the modified project archive when you are finished.

2. Review the source code for the **Lab4Driver** class, which creates a model, two views (windows), and a controller. This is an example of Model, View, Controller (MVC) design pattern. As you can see, all the driver does is create these objects and tell them about one another. After that point, the model, views, and controller interact with one another to do all the work.

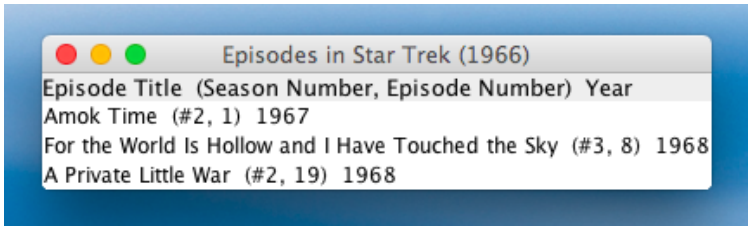
- The fundamental purpose of the model is to encapsulate the data the user cares about and allow the other components to interact with that data in controlled ways. In this program, the data is about episodes of a TV series.
- The two fundamental purposes of the views are (1) to allow the user to provide input to the system and (2) to allow the system to display data to the user. In this program, those two purposes are split into two separate views. In other programs, these purposes may be integrated into a single view or distributed across many views.
- The fundamental purpose of the controller is to provide appropriate links between user input and changes to the model and/or the views.

Some interactions between the model, views, and controller will be through standard method calls. Other interactions will be through "sources" creating "events" and sending them to "listeners."

3. Review the source code for the **Series** and **Episode** classes, which are classes for representing (some aspects of) series and episodes. **Series** is extended by the **SeriesModel** class. This class is the data model for the program. It extends the **Series** class by adding variables and methods, and by overriding methods, in order to deal with the GUI. You will use methods provided in these classes to complete the code for the lab.

4. Review the source code for the **SeriesInputWindow** class, which is a class that presents to the user a GUI window for adding new episodes to the series or for clearing out the episode list. **Label each component of the GUI input window below with the corresponding code variables from SeriesInputWindow.**

5. Review the source code for the **SeriesDisplayWindow** class, which is a class that presents a GUI window to the user to view a series and its associated episode list. **Label each component of the GUI input window below with the corresponding code variables from SeriesDisplayWindow.**



6. Register each object that will serve as an **ActionListener** with its corresponding GUI input component. To determine which objects will serve as **ActionsListener**s, look for classes that implement **ActionListener**. To register an object as an **ActionListener** with its GUI component means that you need to "connect" the **ActionListener** objects to the objects that are listening for events. Do this by following the example shown in the code for `AddEpisodeListener`.

7. Add an `actionPerformed()` method to each class that serves as an **ActionListener** (the classes that implement the **ActionListener** interface). This method should handle the events specified in the source code of **Driver**.

8. Ensure that there are no warnings generated for your code. **Do not suppress warnings.** Fix your code so that warnings are not necessary. (If you can't figure out how to fix your code to avoid the cast warning on the cloned `actionListenerList`, you may leave in that warning.)

9. Submit this file (with completed answers) and the **project archive** following the steps given in the **Submission Instructions** by **11:59 pm on 10 March 2017** through D2L (http://learn.ou.edu).