GUI Lab

Overview: In the Graphical User Interface (GUI) Lab we will construct a simple GUI step by step.

Data Manager: (Provided to the student) It is good programming practice to separate functionality into distinct classes. To accomplish this, GUI functionality is kept separate from the management of the application. The DataManager does not interact with the user. Rather, the GUI collects any user input like mouse-clicks and calls methods of the DataManager, and the DataManager reports results to the GUI which may be displayed to the user. Within the DataManager, there are usually one or more Data Structures, which are populated with one or more types of Data Elements. This DataManager has only simple functionality which does not require data structures other than simple fields.

FXDriver: (Initial version provided to the student) The class called ***FXDriver*** is the starting point for the FX version of the GUI. FX is the framework we will use, and is available beginning with Java 8. There are many ways to organize the GUI, but one way is to hold the basic startup features in a single simple file, which calls an instance of the main panel.

In FX, the ***main*** method (as in any Java application) is the starting point. In FX it calls only the method ***launch(args)***, where ***args*** is the array of data passed in to the application, usually ***null***. This launch method executes various setup code, hidden from the programmer, including creating a so-called "stage", and calls the method ***start(Stage stage)***; Everything that the programmer wants to execute must be placed in ***start***.

FXMainPane: (Initial version provided to the student) In this approach to GUI building, an instance of the class ***FXMainPane*** does all the programmer's work of creating the GUI. The initial version is simply a stub with a constructor, so that FXDriver will compile and run.

Task #1

1) Create a java project in Eclipse. Load the files FXDriver.java, FXMainPane.java, and DataManager.java from Blackboard into this project.
2) In the FXDriver class, create a field of type **FXMainPane** and set it to a new instance of **FXMainPane**() called **root.**
3) Using the parameter called **stage** (provided by Java FX when **launch** calls **start**), call the method **stage.setScene(new Scene(root,** *[width]*,*[height]*));** The parameters width and height specify the dimensions of the scene window in integer pixels.

Compile and run. Note that a window appears, but does not have any content.

Task #2

Note that **FXMainPane** extends **VBox**, so the class is a **VBox**.

1) In the **FXMainPane** class, declare five buttons, a label, and a textfield. Declare two **HBox**es that will hold the other components and go inside the **VBox**.

2) Instantiate the buttons with the arguments “Hello”, “Howdy”, “Chinese“, Clear”, and “Exit”.

3) Instantiate the label with the argument “Feedback:”.

4) Instantiate the textfield.

5) Instantiate the **HBox**es.

Compile and run the **FXDriver**.

Note that even though the buttons and other components are instantiated, they do not appear in the window.

Task #3

To cause components to appear, they must be added to a containing component, which itself is added to its containing component, all the way up to the component that is added to **Scene**.

1) Add the buttons to one of the **HBox**es using **hBox.getChildren().addAll(button1, etc…)**

2) Add the label and the textfield to the other **HBox**

3) Add the **HBox**es to the **FXMainPane** (which is a **VBox**) using **getChildren().addAll(…)**. Note that we have already added **FXMainPane** to the new **Scene** in **FXDriver**.

Compile and run the **FXDriver**.

Note two things: The buttons do not respond to mouse-clicks, and the components are not well-spread, but are all positioned in the upper left corner of the containing window.

Task #4

1) Declare an instance of **DataManager** at the **FXMainPane** class level so that its scope extends throughout the class. Instantiate it in the constructor.

2) To set the buttons to respond to mouse-clicks,

   a) create an inner class at the class level (for example, called ButtonHandler) that implements **EventHandler<ActionEvent>**. For each button, use **button.setOnAction(new ButtonHandler())**.

   b) This interface requires the method **handle(ActionEvent event)** to be implemented. The mouse event is provided to the **handle** method, and **event.getTarget()** returns the object that caused the event, allowing a selection to be made.

   c) Implement an if-else-if structure that gets the target button that was selected and executes the appropriate block of code.
      i) If the "Hello", "Howdy" or "Chinese" buttons were selected, call the **DataManager** instance's methods **getHello**, **getHowdy**, or **getChinese** and call the text field's method **setText** with the response from the **DataManager**.
      ii) If the "Clear" button was selected, call the text field's method **setText("");** to clear out the text field.
      iii) If the "Exit" button was selected, call **Platform.exit();** and **System.exit(0);**

3) Position the components so they are generally centered and spaced apart.

   a) To set margins for each component, call the static method on the containing component, e.g., **HBox.setMargin(button1, inset);** where **inset** is an instance of the class **Inset**.

   b) To center each **HBox**, use **hBox.setAlignment(Pos.CENTER);** where **Pos** is an Enum that will need to be imported.

4) ***JUST FOR FUN (optional)***: create a sixth button that says "Hello" in another language. Add a method to the DataManager class to return the string.

Compile and run FXDriver. Select each of the buttons to ensure they are working correctly.

Submit the three java files to Blackboard.