University of Puerto Rico
Department of Electrical and Computer Engineering

ICOM 4015 Laboratory: Advanced Programming

Laboratory 6: Arrays and Mouse Adapters

Completed by:
ID:
Date:

Introduction

In this laboratory we will practice with arrays that hold color information. Since we will display them graphically, we will use adapters to read the mouse.

What are listeners and adapters?

We use listeners to receive information about user activity, such as mouse movement or keyboard input. Adapters are special cases of listeners: it is slightly easier for us to use adapters than to use listeners. Adapters are *classes* (which we have always used in the laboratory), whereas listeners are *interfaces* (which will be discussed in the future).

The class **MyMouseAdapter** is the adapter we're using to receive and analyze mouse events. We first create a new object of class **MyMouseAdapter** and register it with the frame we're using. This is done by the following two lines within the **Main** class:

```
MyMouseAdapter myMouseAdapter = new MyMouseAdapter();
myFrame.addMouseListener(myMouseAdapter);
```

After those lines are executed, mouse events will be sent to the adapter object. When a mouse button is pressed, the **mousePressed** method gets called. When a button is released, the **mouseReleased** method gets called. There are other events that we can monitor, and those events would call other methods, but we're not using them in this laboratory. If you want to learn more, position the cursor within the *MouseAdapter* identifier (at the top of the **MyMouseAdapter.java** source code) and then press Shift+F2, then take a look at the "Method Summary".

Those methods (**mousePressed** and **mouseReleased**) receive an argument by means of the parameter named **e**, which is of class **MouseEvent**. This means that **e** contains information about the state of the mouse when the relevant event (a mouse press or a mouse released) occurred.

The call **e.getButton()** returns 1 if the event happened because of a change in the left mouse button, 2 if the event happened due to the middle mouse button, 3 due to the right mouse button, and so on. The method calls **e.getX()** and **e.getY()** are used to obtain the mouse position – an (x, y) coordinate – at the time of the press or release. Please note that we use **e.translatePoint** to modify the (x, y) coordinates of the mouse position so that they are based on the insets. That's because we're using insets inside of the **paintComponent** method of **MyPanel**. (Recall that the insets provide us with information about the *interior* region of the frame.)

Exercises

Create a project of name **Lab6** and make it so that it contains the code provided by your instructor.

Run the program and observe what happens when you click on the white cells. Then look at the code to determine what actually happens.

LAB Q1: Describe in your own words how are the squares colored in response to mouse clicks:

//

//

LAB Q2: Sometimes when you click on an *initially white cell*, it does not change color. Modify the code until it always changes color upon each click, then paste the whole **MyMouseAdapter** class inside the slashes below. (Note: This does not apply to cells that are initially gray.) Warning: When comparing color objects, make sure to use .equals(...) instead of ==. (This is the same as with strings!) For example:

```
Color C1 = new Color(0x964B00);   //Brown (simple.wikipedia.org/wiki/List_of_colors)
Color C2 = new Color(0x964B00);   //Brown (simple.wikipedia.org/wiki/List_of_colors)
if (C1.equals(C2)) {
        System.out.println("Yes, they represent the same color.");
}
//Note that C1==C2 would be false, because they are different objects.
```

//
//

LAB Q3: Modify the application so that when the user clicks on one of the initially gray cells on the leftmost column (that is not the top-left cell nor the bottom-left cell), the whole **MyMouseAdapter** row changes color randomly so that no cell of that row stays in the same color. (The leftmost gray cell should stay gray.) When you are done, paste the whole class inside the slashes below.

//
//

LAB Q4: Modify the application so that when the user clicks on one of the initially gray cells at the top (that is not the top-left cell), the whole column changes color randomly so that no cell of that column stays in the same color. (The topmost gray cell should stay gray.) When you are done, paste the whole **MyMouseAdapter** class inside the slashes below.

//
//

LAB Q5: Modify the application so that when the user clicks on the top-left gray cell, the whole main diagonal (the one that runs from the top-left to the bottom-right) changes color randomly so that no cell of that diagonal stays in the same color. (The top-left cell should stay gray.) When you are done, paste the whole **MyMouseAdapter** class inside the slashes below.

//
//

LAB Q6: Modify the application that clicking on the bottom-left gray cell changes the nine cells at the *center* of the inner grid (that's a 3*x*3 region) so that none of those cells stays in the same color. When you are done, paste the whole **MyMouseAdapter** class inside the slashes below.

//
//

LAB Q7: Modify the application so that pressing the right mouse button anywhere outside of the grid will change all of the non-gray cells to any of three random colors that you choose, that are not any of the seven colors already in use by the program. When you are done, paste the whole **MyMouseAdapter** class inside the slashes below. **Note:** You may find it useful to invoke the **MyPanel.getGridX** and **MyPanel.getGridY** methods that have been provided.

//
//