ITI 1121. Introduction to Computer Science II

Laboratory 5

Summer 2015

Plan

- Introduction to Graphical User Interfaces (GUIs)
- Exercises related graphical user interfaces

This document contains links towards the solutions of the exercises, please make genuine efforts at answering the questions before looking at the solutions!

Concepts: event-driven programming

- Any component can be the source of an event; a button generates an event when you click on it;
- Any class can be a listener for an event; it simply implements the method(s) of an interface.
  The object that handles the event generated by the click of a button needs to implement the interface **ActionListener**.

  The object that handles the event generated when a window is closed must implement the interface **WindowListener**;
- The event generated by the source component is sent to all the listeners who registered with the component.
  The source has a method **addActionListener** who needs as a parameter, an object that implements the interface **ActionListener**.

  Since the handler implements the interface **ActionListener**, the source knows that the handler has a method **actionPerformed(ActionEvent event)**.

1 Moving the red dot

Here is a simple application that has a display area and two buttons. When the left button is clicked the dot moves to the left; similarly for the right button. *Adapted from Decker & Hirshfield (2000) Programming.java*

1.1 Creating the graphical layout of the application

1.1.1 DisplayArea

Create a subclass of **Canvas**, called **DisplayArea**. "A Canvas component represents a blank rectangular area of the screen onto which the application can draw (…)". Later, we'll add the necessary implementation details so that a red dot is displayed in the canvas and moved to the left or to the right whenever the user

clicks on the left or the right button. Add a constructor that sets the size of the **DisplayArea** to 200 by 200 pixels.

1.1.2 GUI

Create a subclass of **Frame**, called **GUI**. This will be the main window of the application.

- Add an instance variable of type **Button** and initialize this with an object labeled “Left”;
- Add an instance variable of type **Button** and initialize this with an object labeled “Right”;
- Add an instance variable of type **DisplayArea**.

Create a constructor that will add all the necessary graphical components to create the layout.

- Changing the background color, **setBackground(Color.WHITE)**;
- Add the **DisplayArea** object to the center of the **Frame**;
- Create a **Panel**;
- Add the left and right buttons to the **Panel**;
- Add the panel to the **SOUTH** region of the **Frame**;
- Add a main method that simply creates an instance of the class **GUI**.

There are two ways to start this application. In a shell (dos/command window) type:

```
> java GUI
```

Or create a new instance of **GUI** in the interactions window of **DrJava**:

```
> GUI f = new GUI();
```

Your application should look something like this.

Clicking the closing button of the title bar will have no effect, this is because we have not associated any action with the button yet. If you are using **DrJava**, then you can make the window invisible by typing the following statement.

```
> f.setVisible( false );
```

Solutions

- DisplayArea.java
- GUI.java

1.2 MyWindowAdapter

Clicking on the closing button of the title bar has no effect. Our main window is an object of the class **GUI** (a **Frame**). A frame has a method **addWindowListener**. It allows to register objects that will asked to perform the required actions when the user generates such event. The method **addWindowListener** requires an object that implements the interface **WindowListener**. The interface lists the following 7 methods.

```
void    windowActivated(WindowEvent e)
         Invoked when the Window is set to be the active Window.

void    windowClosed(WindowEvent e)
         Invoked when a window has been closed as the result of
         calling dispose on the window.

void    windowClosing(WindowEvent e)
         Invoked when the user attempts to close the window from
         the window's system menu.

void    windowDeactivated(WindowEvent e)
         Invoked when a Window is no longer the active Window.

void    windowDeiconified(WindowEvent e)
         Invoked when a window is changed from a minimized to
         a normal state.

void    windowIconified(WindowEvent e)
         Invoked when a window is changed from a normal to a
         minimized state.

void    windowOpened(WindowEvent e)
         Invoked the first time a window is made visible.
```

The relevant method here is **windowClosing**, when the user clicks on the window closing button, your application should call **System.exit(0)**. It is not possible to implement a single method, you are forced to provide an implementation for all the methods listed in the interface. You could provide empty definitions for all the other methods but in fact this is such a commonly occurring scenario that the makers of Java created the following abstract class, called an adapter:

```
public abstract class WindowAdapter implements WindowListener {
    public void windowOpened( WindowEvent e ) {}
    public void windowClosing( WindowEvent e ) {}
    public void windowClosed( WindowEvent e ) {}
    public void windowIconified( WindowEvent e ) {}
    public void windowDeiconified( WindowEvent e ) {}
    public void windowActivated( WindowEvent e ) {}
```

```
    public void windowDeactivated( WindowEvent e ) {}
}
```

It provides an empty definition for all the methods of the interface. Whenever you want to create a window listener that only implements a subset of the methods of the interface, you simply have to create a subclass of WindowAdapter and override the relevant methods.

- Create a subclass of WindowAdapter, called MyWindowAdapter;
- Override the method windowClosing(WindowEvent e). The implementation calls System.exit(0);
- Modify the constructor of the class GUI adding the following statement.
  `addWindowListener( new MyWindowAdapter() );`

Compile and test the application. You should now be able to exit the application by clicking the closing button.

Solutions

- DisplayArea.java
- GUI.java
- MyWindowAdapter.java

1.3 Action listeners 1/2

Similarly, we need to tell the application how to handle the user events. For this, we need to modify the constructor of the class GUI and add an ActionListener to each button. Which class will be responsible for this? When a button is pressed, the application needs to re-draw a red circle onto the canvas, because of this let's modify the canvas so that it becomes the ActionListener for the two buttons.

- Modify DisplayArea so that it implements ActionListener. This requires implementing a method called actionPerformed(ActionEvent e);
- For now, the implementation of actionPerformed simply displays a message, e.g. "actionPerformed was called";
- Modify the constructor of the class GUI, for each button call the method addActionListener to register the object DisplayArea.

You can now compile and test the application. Each time one of the two buttons is pressed, a message will be printed into the console.

Solutions

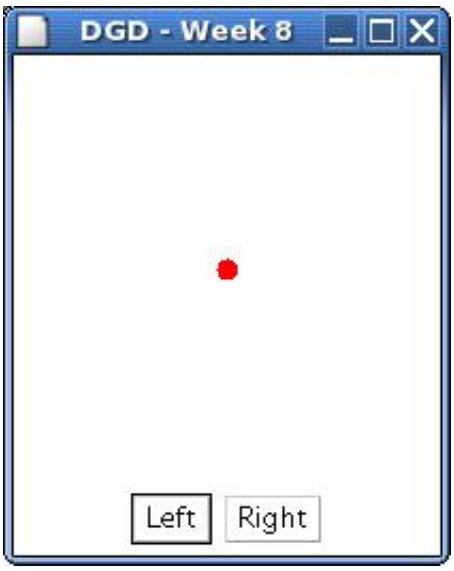
- DisplayArea.java
- GUI.java
- MyWindowAdapter.java

1.4 paint

Each time AWT needs to paint a Canvas it calls its method paint(Graphics g).

- Modify DisplayArea, add an instance variable of type Point called center;
- Modify the constructor and initialize center to “new Point(100, 100)”;
- Override the method paint(Graphics g). You will be painting onto the object designated by g. First set the default color to red, g.setColor(Color.RED), then paint a circle, of diameter 10, centered at center.x, center.y, g.fillOval(center.x - 5, center.y - 5, 10, 10).

You can now compile and test the application. You should see a red circle at the centre of the display area.



Solutions

- DisplayArea.java
- GUI.java
- MyWindowAdapter.java

1.5 Action listeners 2/2

All that is left to do is making the dot move, to the left or to the right, when the user clicks onto the left or right button.

- This requires modifying the method actionPerformed of the class DisplayArea. First it needs to determine which button was pressed, fortunately, the method getActionCommand of the ActionEvent returns the String that was used when creating the buttons, “Left” or “Right”;
- If getActionCommand returns “Left”, then we will move the centre 10 pixels to the left;
- If getActionCommand returns “Right”, then we will move the centre 10 pixels to the right;
- We now want to force a call to our method paint(Graphics g), this can be done by calling the method repaint().

You can now compile and test the application.

Solutions

- DisplayArea.java
- GUI.java
- MyWindowAdapter.java

2 Quiz (1 mark)

- Draw memory diagrams for all variables and objects created during the execution of the main method below.

```
class Singleton {
    private int value;
    Singleton( int value ) {
        this.value = value;
    }
}

class Pair {
    private Object first;
    private Object second;
    Pair( Object first, Object second ) {
        this.first = first;
        this.second = second;
    }
}

public class Quiz {
    public static void main( String[] args ) {
        Singleton s;
        Pair p1, p2;

        s = new Singleton( 99 );
        p2 = new Pair( s, null );
        p1 = new Pair( s, p2 );
    }
}
```

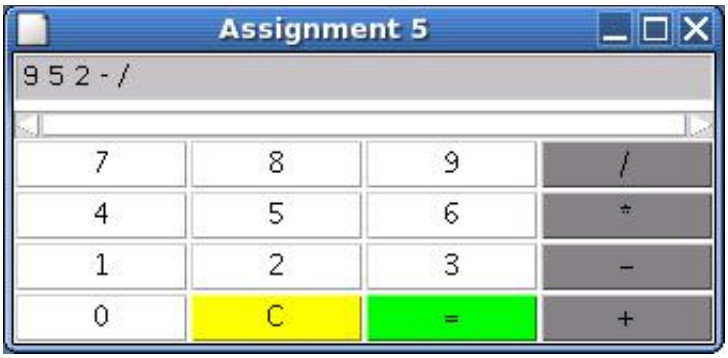
You can either use a pencil and paper approach or use a drawing/painting program. If you are drawing on a piece of paper, then you may take a picture and upload the picture to blackboard. If this is not possible, then you have to submit a texfile that states the drawing will be handed in to the TA and give your drawing to your TA who will record the information (your mark for the quiz). You need to upload a file (either the diagram or text file explanation in order to get a mark).

- https://uottawa.blackboard.com/

3 Calculator (Optional)

Here is one more example to help you better understand graphical user interfaces. You can work on this example during the laboratory, if time allows, or take it home. The solutions are provided.

Follow the step-by-step approach proposed for the red dot application, and create an RPN calculator as displayed here.



3.1 Create the graphical layout of the application

Call this class Calculator. The display region is a TextArea. I used a panel with a GridLayout manager, 4 by 4 cells, to create the key panel.

3.2 WindowListener

Make sure to add a WindowListener that implements the windowClosing method.

3.3 ActionListener

Which class will be responsible for the handling the user generated events? It could be Calculator. How to determine which button was clicked? Use the class RPN below to implement the actions.

Solutions

- Calculator.java
- RPN.java

Last Modified: June 22, 2015