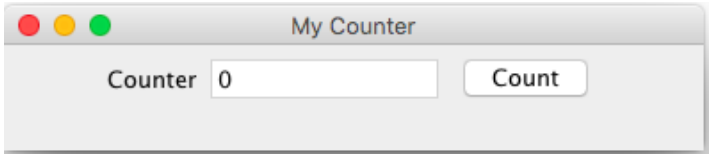
Tutorial - Week 09 5COSC019W – Object Oriented Programming – Java

Event Listener, File Handling

21-22 November

GUI application

1) Write a Swing GUI application called Counter as shown in figure below. Every time the Count button is clicked, the counter value should increase by 1.



The user interface has 3 JComponent:

- JLabel
- JButton
- JTextField

Swing Components are to be added onto the ContentPane of the top-level container JFrame. You can retrieve the ContentPane via method getContentPane() from a JFrame.

Note: Swing Components are kept in package javax.swing. They begin with a prefix "J", e.g., JButton, JLabel, JFrame.

```
import java.awt.*;       // Using AWT's layouts
import javax.swing.*;    // Using Swing components and containers
```

Write a class Counter that extends JFrame:

```
public class Counter extends JFrame{

   private JLabel lblCount;      // Declare component JLabel
   private JTextField tfCount;   // Declare component JTextField
   private JButton btnCount;     // Declare component JButton
   private int count = 0;       // counter's value

    // Constructor to setup UI components and event handlers
   public Counter () {
       // sets layout to FlowLayout, which arranges
       // Components from left-to-right, then top-to-bottom.
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());

      lblCount = new JLabel("Counter"); // Construct component Label
      add(lblCount);                    // "super" Frame adds Label

      tfCount = new JTextField(count + "", 10); // Construct component TextField
      tfCount.setEditable(false);               // read-only
      add(tfCount);                             // "super" Frame adds TextField

      btnCount = new JButton("Count");   // Construct component Button
      add(btnCount);                     // "super" Frame adds Button
   }
```

Instance variables

Implement Constructor

Get content Pane and set the layout

Add compo-nent to the content pane

Event Handling

The ***first step*** is to write an inner class that `implements EventListener`.

- An inner class called MyListener is defined, to be used as listener for the ActionEvent fired by the Button btnCount. Since MyListener is an ActionEvent listener, it has to implement ActionListener interface and provide implementation for the `actionPerformed()` method declared in the interface.

- Even if instance variables `tfCount`,and `count` are private, the inner class `MyListener` has access to them. This is the sole reason why an inner class is used instead of an ordinary outer class.

In the same file write:

```
private class MyListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent evt) {
         ++count;
         tfCount.setText(count + "");
      }
   }
```

Every time an ActionEvent is generated the count variable is incremented by 1 and the value is displayed in the GUI

The ***second step*** is to register the listener to the JComponent: in your code you need to add the following line of code in the Counter constructur

```
MyListener handler = new MyListener();
btnCount.addActionListener(handler);
```

- btnCount is the **source** object that fires ActionEvent when clicked
- The source add "handler" instance as an ActionEvent listener, which provides an ActionEvent handler called actionPerformed().
- Clicking btnCount invokes actionPerformed().

2) Modify the previous code in order to include two additional buttons for counting down and reset the count value. Hints:

- Add other two JButtons to the content pane and register the Event Listener
- You can use `event.getActionCommand()` to retrieve the label of the button that has fired the event

```
public void actionPerformed(ActionEvent evt) {
          String btnLabel = evt.getActionCommand();
       // event.getActionCommand() returns the button's label
        if (btnLabel.equals("Count Up")) {
            //…

       } else if (btnLabel.equals("Count down")) {
           //…
       } else if (btnLabel.equals("Reset")){
          //…
      }
      }
   }
```

File Handling

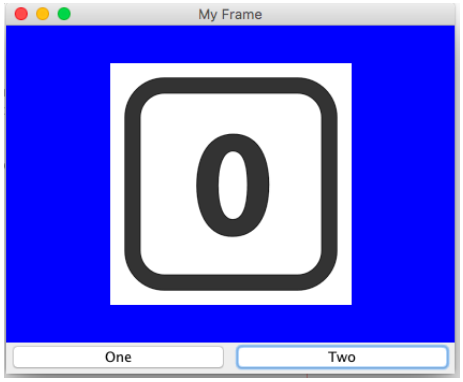
3) Create a class and the necessary methods to read name, id and mark of a student from the console and save it in a file. Use `FileReader` and `FileWriter`, `BufferedReader` and `BufferedWriter` to write a program that has the following functional menu:

```
Menu
1 - Add Student in the List and save to file
2 - Read Students List from a file and display on the screen
3 - Exit
Enter your choice:
```

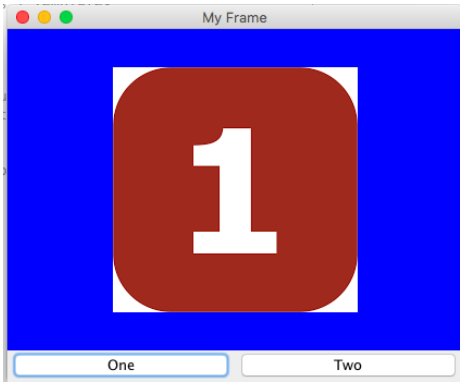
Show Images

Write a Swing GUI application that shows three different images (download the images from BB). The Jframe has the following components:

- Jlabel (the image is displayed on this label)
- Two JButtons

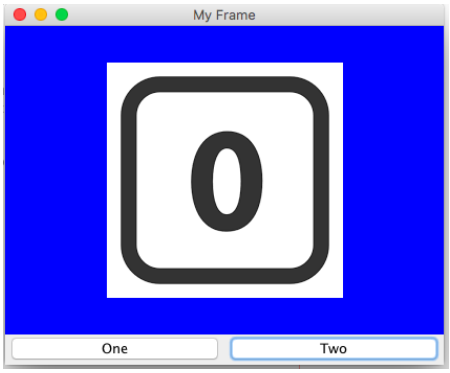


a) When you **run the application** you will display a default image that represents a zero.



b) When you **click on "One" button** the image will change and it will display a one

c) When you **click on "Two" button** the image will change and it will display a two



d) When you **click on the image** it will return to display the default zero image.

Solution Show images:

Implement a class called `ShowImageFrame` that extends `JFrame`. Write the instance variables and the constructor for this class. You need to:

- Set the layout in order to have the JLabel in the center and the two buttons on the bottom
- Set a default image to the JLabel
- Add listeners to your JComponents

```
public class ShowImageFrame extends JFrame{

    JButton btnOne;
    JButton btnTwo;
    JLabel lblImage;

    ImageIcon imageIconZero;
    ImageIcon imageIconOne;

    ImageIcon imageIconTwo;

    public ShowImageFrame(){

      Container cp = getContentPane();
      // set layout in the main panel
      cp.setLayout(new BorderLayout());
      cp.setBackground(Color.blue);

      // create a panel with two buttons in a grid
      JPanel p1 = new JPanel();
      p1.setLayout(new GridLayout(1,2));
      //create and add buttons
      btnOne = new JButton("One");
      p1.add(btnOne);

      btnTwo = new JButton("Two");
      p1.add(btnTwo);

      // create a label with a default image
      imageIconZero = new ImageIcon(getClass().getResource("zero.png"));
      lblImage = new JLabel(imageIconZero, JLabel.CENTER);

      //add the jlabel on the top
      cp.add(lblImage, BorderLayout.CENTER);

      //add the button on the bottom
      cp.add(p1, BorderLayout.SOUTH);

      // instantiate the other two images
      imageIconOne = new ImageIcon(getClass().getResource("one.png"));
      imageIconTwo = new ImageIcon(getClass().getResource("two.png"));

      // add event handler
      MyListener myListener = new MyListener();
      btnOne.addActionListener(myListener);
      btnTwo.addActionListener(myListener);

      // add mouse listener to the jLabel
      lblImage.addMouseListener(myListener);

    }
```

Instance variables

We represent our images with an `ImageIcon` object.
(`ImageIcon` class paints Icons from Images)

Constructor

- We retrieve the image reading the png file from the resource folder and we save it as `ImageIcon` object.
- We pass the `ImageIcon` object as argument to JLabel constructor

- We add an action listener to button One and button Two
- We add a mouse listener to the JLabel

1) Implement a listener to handle the `ActionEvent` and the `MouseEvent`. Read the following code and explain why `MyListener` extends `MouseAdapter` and implements `ActionListener`.

```
private class MyListener extends MouseAdapter  implements ActionListener{

      public void actionPerformed(ActionEvent evt) {
          String btnLabel = evt.getActionCommand();

       // event.getActionCommand() returns the button's label
        if (btnLabel.equals("One")) {
            lblImage.setIcon(imageIconOne);

        } else if (btnLabel.equals("Two")) {
            lblImage.setIcon(imageIconTwo);
        }
    }

      public void mouseClicked(MouseEvent evt){
          lblImage.setIcon(imageIconZero);
      }
}
```

2) Write the main class to display your ShowImageFrame:

```
public static void main(String[] args)  {
      // Invoke the constructor by allocating an anonymous instance
      ShowImageFrame myFrame = new ShowImageFrame();

      myFrame.setSize(600, 400);       // "super" Frame sets initial size
      myFrame.setTitle("My Frame");           // "super" Frame sets title
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setVisible(true);                  // show "super" Frame
}
```