Object Oriented and Java Programming

Course 6 –GUI design[1]

Qiong Liu

qiong.liu@cyu.fr
CY Tech, CY Cergy Paris University

October 2024

[1] Thanks to the tutorial of Dr. Wang S.S.

Contents

Outline

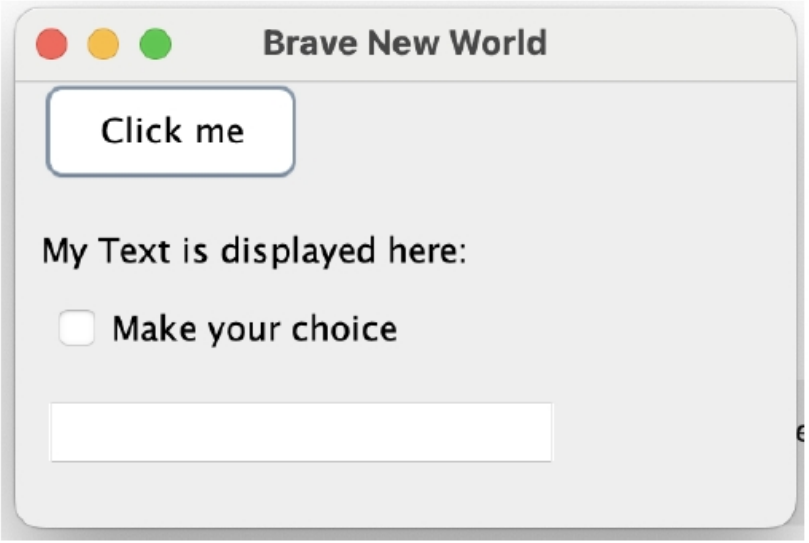
What is Swing

Definition

Swing is a light GUI (graphical user interface) toolkit created for Java based applications.

It contains multiple components:

- window
- text field
- label
- checkbox
- button
- ...



History on Swing

Java's original GUI library was the Abstract Window Toolkit (AWT). Swing was added to the platform in Java SE 1.2 (1998). Since then, Swing has remained the primary Java GUI technology.

AWT:

- AWT is Java's original toolkit for GUI.
- AWT is heavyweight means it uses underlying operating system to display the components.
- AWT is platform specific.

However,

- Heavyweight $\rightarrow$ slow in running
- platform specific $\rightarrow$ no identical looking in different OS

History on Swing

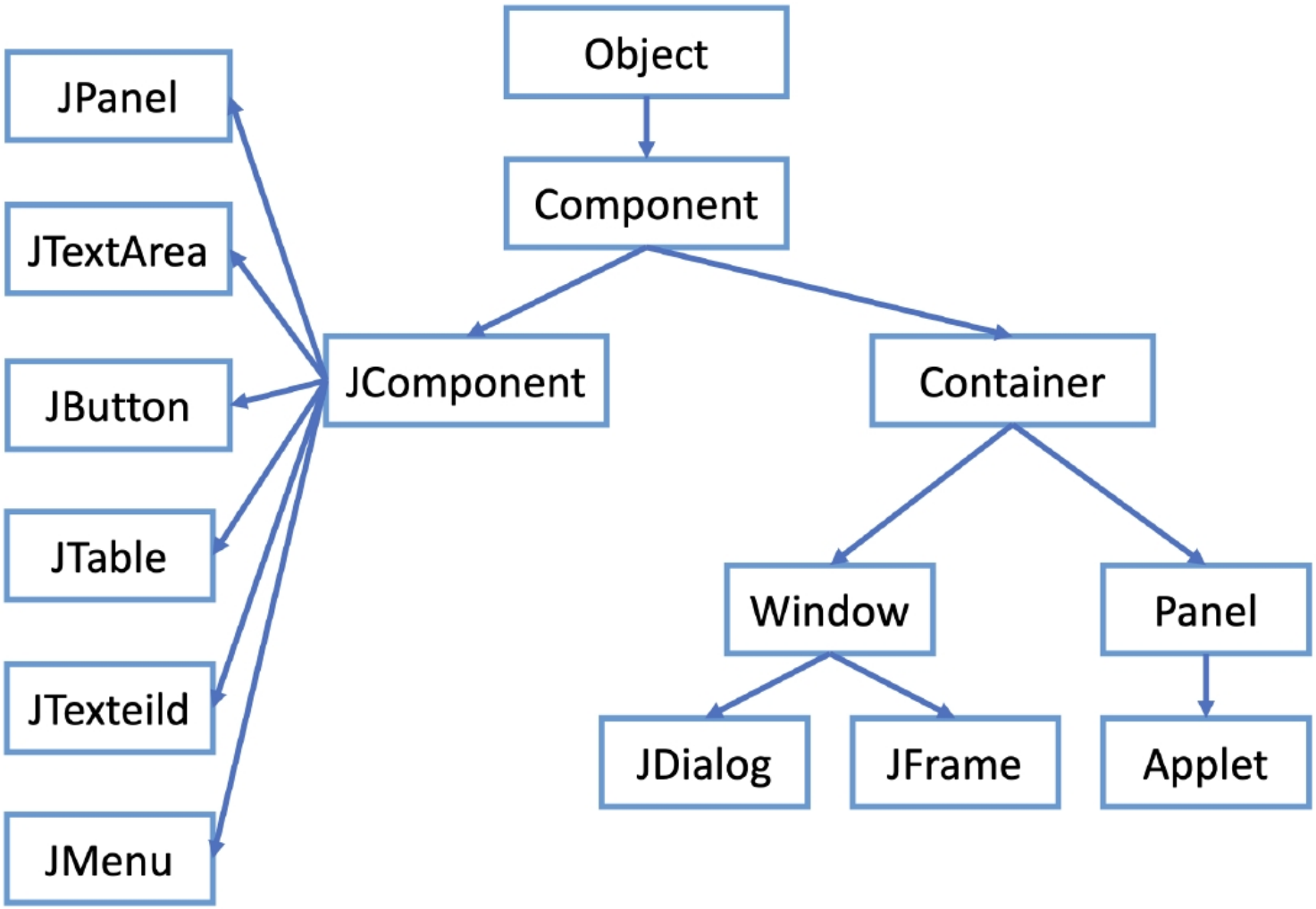
Swing:

- Swing is a lightweight toolkit. It means that it is entirely written in Java. So it runs fast.
- Swing is built on top of AWT packages.
- Platform independent.
- It is part of Java Foundation Classes (JFC), and includes several packages for developing rich desktop applications in Java.

Current Status: Swing is now in maintenance mode—Oracle has stopped development and will provide only bug fixes going forward.
Instead, JavaFX is brought up, as Java's new toolkit for GUI since 2008.

Outline

Structure


Object
Component
JPanel
JTextArea
JButton
JTable
JTexteild
JMenu
JComponent
Container
Window
Panel
JDialog
JFrame
Applet

JFrame

JFrame is used to create the window in a GUI application.

```
public class TestWindow {
    public static void main(String[] args){
        JFrame window = new JFrame("New World");
        // window.setTitle("New World");
        window.setBounds(500, 200, 700, 550);
        // window.setLocation(500, 200);
        // window.setSize(700, 550);
    }
}
```

To display the window:

```
window.setVisible(true);
```

To have flow layout of the window:

```
window.setLayout(new FlowLayout());
```

JFrame

To close the window:

```
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Options for setDefaultCloseOperation

- EXIT_ON_CLOSE
- HIDE_ON_CLOSE
- DISPOSE_ON_CLOSE
- DO_NOTHING_ON_CLOSE

To allow flexible window size by dragging:

```
window.setResizable(true);
```

JDialog

JDialog has similar functionality as JFrame:

- `JDialog()`: creates an empty dialog without any title or any specified owner.
- `JDialog(Frame o)`: creates an empty dialog with a specified frame as its owner.
- `JDialog(Frame o, String s)`: creates an empty dialog with a specified frame as its owner.
- JDialog can be attached to Window and Dialog as well.

Example Code:

```
JDialog d = new JDialog(window, "Dialog Box");
JLabel l = new JLabel("this is a dialog box");
d.add(l);
d.setSize(100, 100);
d.setVisible(true);
```

JCheckBox and JTextField

JCheckBox:

```
JCheckBox checkbox = new JCheckBox("Make your choice")
    ;
checkbox.setBounds(10, 70, 200, 50);
window.add(checkbox);
```

JTextField:

```
JTextField tfield = new JTextField(20);
tfield.setBounds(10, 120, 200, 30);
tfield.getText();
window.add(tfield);
```

JButton

Creating and adding a JButton:

```
JButton button = new JButton("Click me");
button.setBounds(10, 0, 100, 40);
window.add(button);
```

Enable and disable the button:

```
button.setEnabled(true); // Enable the button
button.setEnabled(false); // Disable the button
```

Note: Icons can be added to button components using the same methods.

JLabel

Creating and adding a JLabel:

```
JLabel label = new JLabel("My Text is displayed here:");
label.setBounds(10, 40, 200, 50);
window.add(label);
```

To add an icon to the label:

```
ImageIcon image = new ImageIcon("test.jpg");
JLabel label = new JLabel("My Text is displayed here:");
// Resize the image
Image originalImage = image.getImage();
int width = 50;
int height = 20;
Image resizedImage = originalImage.getScaledInstance(width, height,
    Image.SCALE_SMOOTH);
ImageIcon resizedIcon = new ImageIcon(resizedImage);
label.setIcon(resizedIcon);
window.add(label);
```

Exercise 1

To set the color and font for labels, you can use method `setForeground()` from `JLabel`, `Font()` class. For example:

```
JLabel cl = new JLabel("test");
Font font = new Font(Font.MONOSPACED, Font.ITALIC, 10)
    ;
cl.setFont(font);
cl.setForeground(Color.BLUE); // add RGB: new Color
    (1,1,1)
cl.setBounds(10, 40, 200, 50);
window.add(cl);
```

Exercise: Please repeat the tutorials previously.

Exercise 1

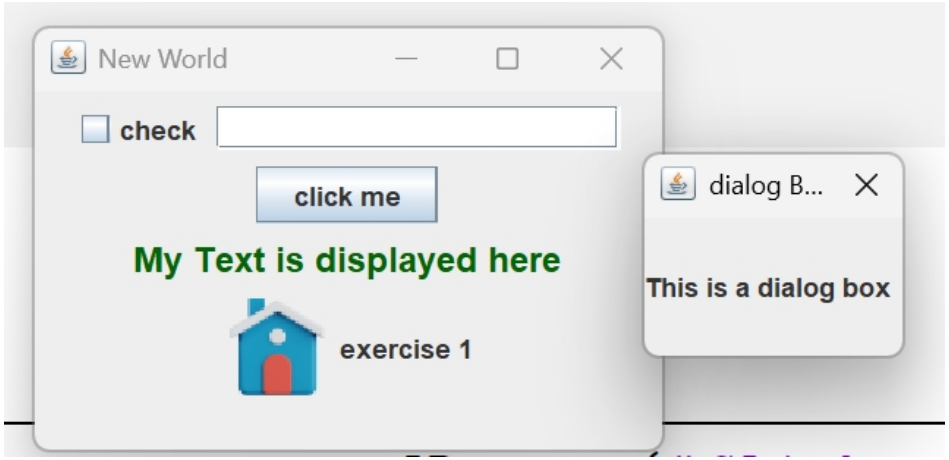


Figure: Exercise 1

- Creating the frame
- JDialog
- JCheckBox
- JTextField
- JButton
- Creating the label with initial text
- Add an icon

JPanel with FlowLayout

```
// Create a JPanel with FlowLayout
JPanel panel1 = new JPanel(new FlowLayout(FlowLayout.CENTER, 10,
    50));// the horizontal and vertical gap between components is
    10 and 50
panel1.setBackground(Color.blue);

// Create two button components
JButton button1 = new JButton("Button a");
JButton button2 = new JButton("Button b");

// Add buttons to the panel
panel1.add(button1);
panel1.add(button2);

// Add panel to the window at the bottom
window.add(panel1, BorderLayout.SOUTH);
```

- The JPanel uses a FlowLayout that centers components and sets gaps between them.
- Buttons are added to the panel which is then added to the south region of the frame using BorderLayout.

JScrollPane

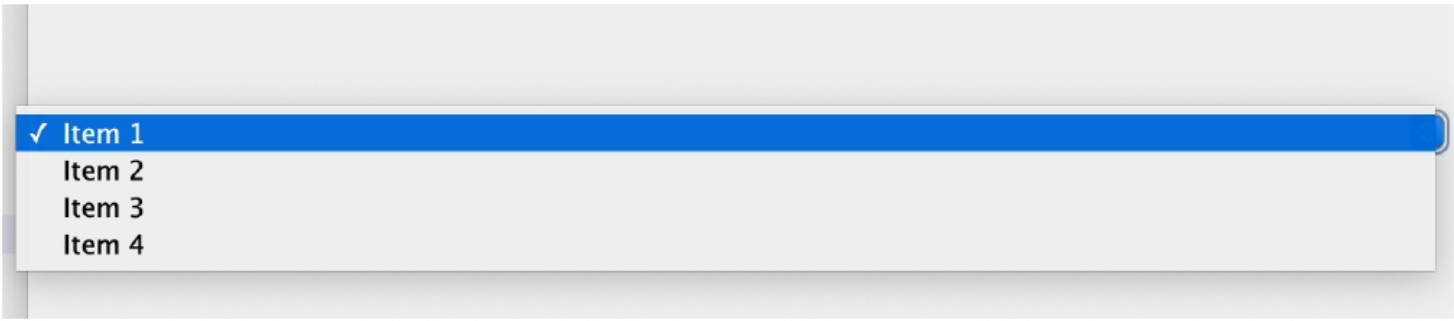
A JScrollPane is used to make a scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

- Only one component can be added to JScrollPane.
- In case of multiple components, components must be added to a JPanel first, then JPanel is added to JScrollPane.

```
// Set the layout for the window
window.setLayout(new FlowLayout());

// Create a JTextArea
JTextArea textArea = new JTextArea(5, 5);

// Create a JScrollPane and add JTextArea to it
JScrollPane scrollPane = new JScrollPane(textArea);

// Add the JScrollPane to the window
window.add(scrollPane);
```

JComboBox

```
// Create a new JComboBox
JComboBox<String> cmb = new JComboBox<>();
// Add items to JComboBox
cmb.addItem("Item 1");
cmb.addItem("Item 2");
cmb.addItem("Item 3");
cmb.addItem("Item 4");

window.add(cmb);
```



Menu

- JMenuBar
  - JMenu
    - JMenuItem

```
JMenuBar bar = new JMenuBar();
JMenu m = new JMenu("Menu 1");
JMenuItem it1 = new JMenuItem("Item 1");
JMenuItem it2 = new JMenuItem("Item 2");
JMenuItem it3 = new JMenuItem("Item 3");

m.add(it1);
m.add(it2);
m.add(it3);
bar.add(m);

window.setJMenuBar(bar);
```

Exercise 2:

Write code in Java to display this window:

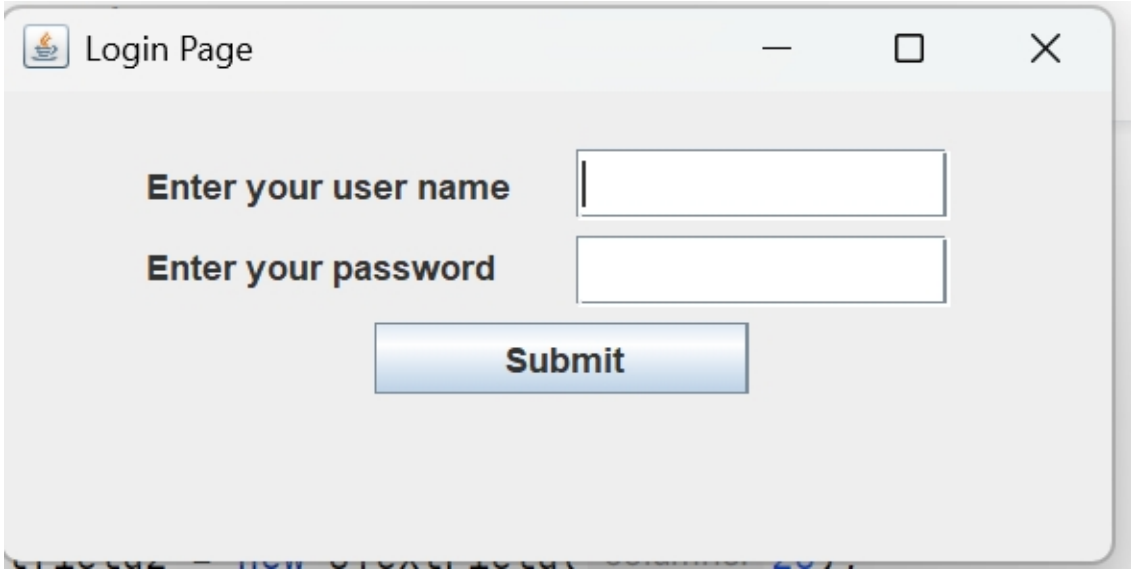


Figure: Caption

Exercise 2: Solution

```
public static void main(String[] args) {
    JFrame window = new JFrame("Login Page");
    window.setBounds(200, 200, 400, 200);
    window.setLayout(null);

    JLabel label1 = new JLabel("Enter your user name")
        ;
    label1.setBounds(50, 20, 150, 25);
    window.add(label1);

    JTextField textField = new JTextField(20);
    textField.setBounds(200, 20, 130, 25);
    window.add(textField);

    JLabel label2 = new JLabel("Enter your password");
    label2.setBounds(50, 50, 150, 20);
    window.add(label2);
```

Exercise 2: Solution (continued)

```
// Add a second text field for password input
JTextField tfield2 = new JTextField(20);
tfield2.setBounds(200, 50, 130, 25);
window.add(tfield2);

// Add a submit button
JButton jb = new JButton("Submit");
jb.setBounds(130, 80, 130, 25);
window.add(jb);

// Set JFrame properties
window.setVisible(true);
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window.setResizable(true);
```

Create frames in a class

When you are creating long program. You should not put everything in main method. It will cause the program having less abstraction, and hierarchy.
We do not want all the details to be exposed.
Therefore we use inheritance to write the class for GUI.

Create frames in a class

```
import javax.swing.*;
import java.awt.*;

public class NewFrame extends JFrame {
    public NewFrame(String title) {
        super(title);
        this.setBounds(100, 100, 500, 300);
        this.setLayout(new FlowLayout());
    }

    public static void main(String[] args) {
        NewFrame jf = new NewFrame("My Window");
        JLabel cl = new JLabel("ORANGE");
        cl.setForeground(Color.BLUE);
        jf.add(cl);
        jf.setVisible(true);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Outline

ActionListener

- `ActionListener` is used to respond to button clicks.
- The `actionPerformed` method is invoked whenever the button is clicked.
- This setup demonstrates creating a simple GUI with an interactive button.

Implementing ActionListener in Java Swing

```
public class NewFrame extends JFrame {
    public NewFrame(String title) {
        super(title);
        this.setBounds(100, 100, 500, 300);
        this.setLayout(new FlowLayout());

        JButton jb = new JButton("Click me");
        myactionL al = new myactionL();
        jb.addActionListener(al);
        this.add(jb);

        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    private class myactionL implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println("Button is clicked");
        }
    }
    public static void main(String[] args) {
        new NewFrame("Action Listener Example");
    }
```

ActionListener

Create an ActionListener that prints the current time when a button is clicked.

```
private class myactionL implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button is clicked");
        SimpleDateFormat sdf = new SimpleDateFormat("
            HH:mm:ss");
        String timeStr = sdf.format(new Date());
        System.out.println("Button is clicked at " +
            timeStr);
    }
}
```

ActionListener

Create an ActionListener that prints the current time when a button is clicked.

```
private class myactionL implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button is clicked");
        SimpleDateFormat sdf = new SimpleDateFormat("
            HH:mm:ss");
        String timeStr = sdf.format(new Date());
        System.out.println("Button is clicked at " +
            timeStr);
    }
}
```

What if we want the current time to be displayed on the GUI?

ActionListener

We need ActionListener to be able to modify JLabel. However, it is not accessible by internal class myactionL.
Solution: Move JLabel out of the method.

```
public class NewFrame extends JFrame {
    JLabel cl = new JLabel("Current Time is: 00:00:00"
        );

    public NewFrame(String title) {
        ...
    }
}
```

And, add following in the class myactionL:

```
cl.setText("Current Time is: " + timeStr);
```

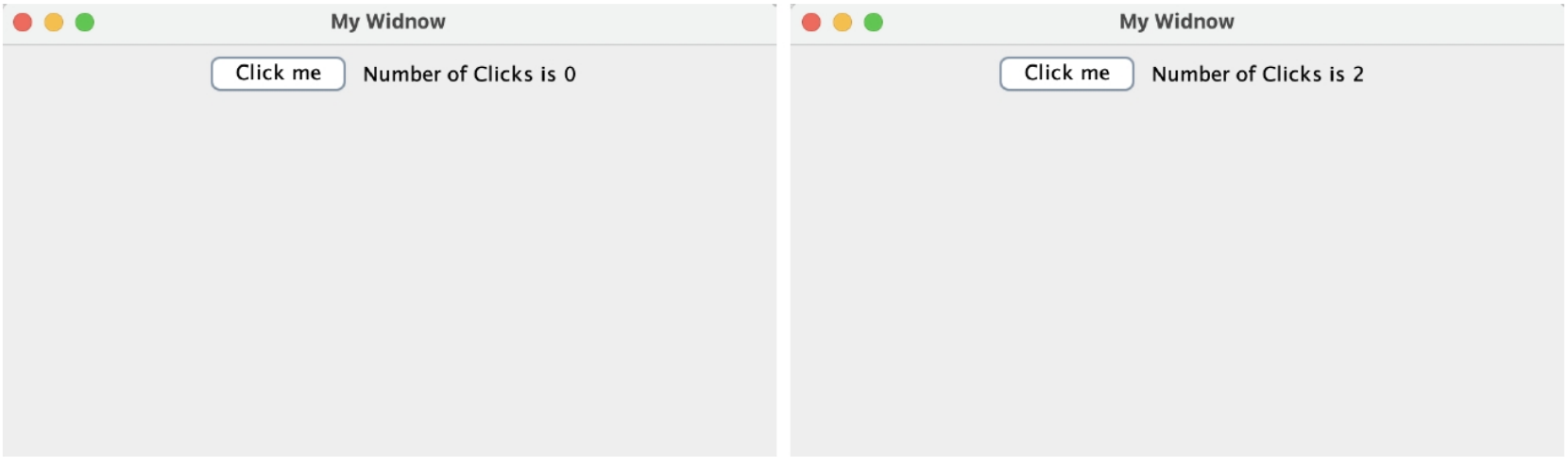
Note: Here `cl.setText()` is short for `NewFrame.this.cl.setText()`.

Exercise 3

Create a JLable. It can display the number of times of clicks.

Exercise 3

Create a JLable. It can display the number of times of clicks.




```
public class NewFrame extends JFrame {
    JLabel cl = new JLabel (" Number of Clicks is 0");
    int num = 0;
...
public void actionPerformed ( ActionEvent e) {
    num ++;
    cl. setText (" Number of Clicks is "+ num);
```

Outline

Layout Options in Javax.swing

- The placement of components on a form is controlled by *layout managers*, not by absolute positioning.
- Layout managers decide the arrangement based on the order components are added (`add()` method).

Behavior Across Managers:

- The size, shape, and placement of components can vary significantly across different layout managers.
- Layout managers adapt to changes in window dimensions, affecting the size, shape, and placement of components.

Components and Containers:

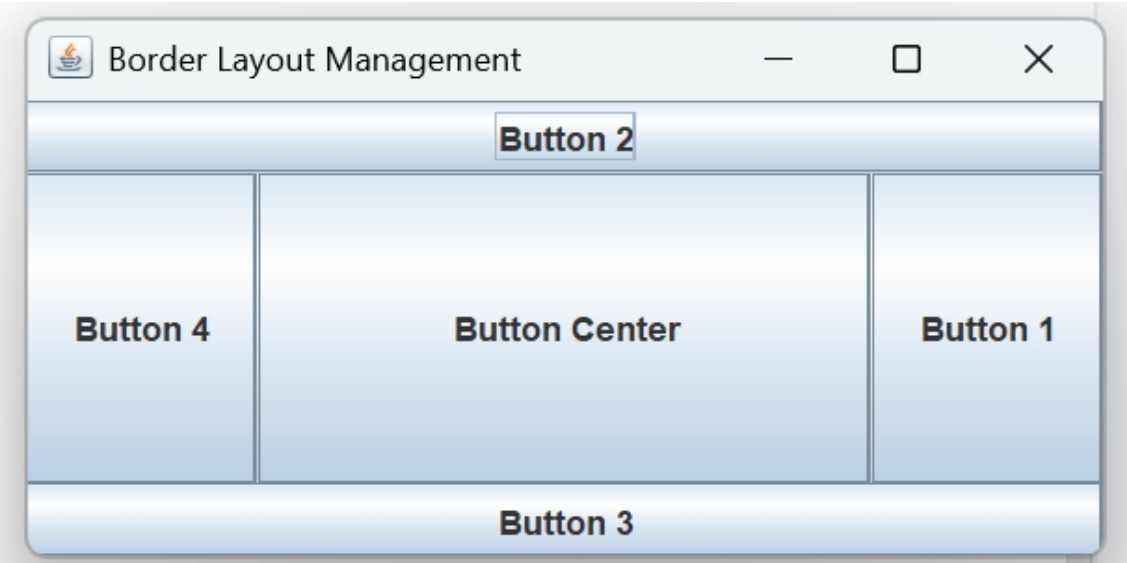
- Common containers such as JApplet, JFrame, JWindow, JDialog, JPanel support different layout managers.
- Use the `setLayout()` method to choose a different layout manager for these containers.

BorderLayout

BorderLayout Overview:

- **BorderLayout.NORTH** - Top of the container
- **BorderLayout.SOUTH** - Bottom of the container
- **BorderLayout.EAST** - Right side of the container
- **BorderLayout.WEST** - Left side of the container
- **BorderLayout.CENTER** - Center space, filling remaining area

If no specific area is designated for a component, it defaults to **BorderLayout.CENTER**.



BorderLayout

```
public class LayoutFrameBorder extends JFrame {
    public LayoutFrameBorder(String title) {
        super(title);
        this.setBounds(100, 100, 400, 200);
        this.setLayout(new BorderLayout());

        // Adding buttons to each region of the BorderLayout
        JButton jb1 = new JButton("Button 1");
        this.add(jb1, BorderLayout.EAST);

        JButton jb2 = new JButton("Button 2");
        this.add(jb2, BorderLayout.NORTH);

        JButton jb3 = new JButton("Button 3");
        this.add(jb3, BorderLayout.SOUTH);

        JButton jb4 = new JButton("Button 4");
        this.add(jb4, BorderLayout.WEST);

        // Adding a button to the center region
        JButton jbCenter = new JButton("Button Center");
        this.add(jbCenter, BorderLayout.CENTER);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new LayoutFrameBorder("Border Layout Management");
    }
}
```

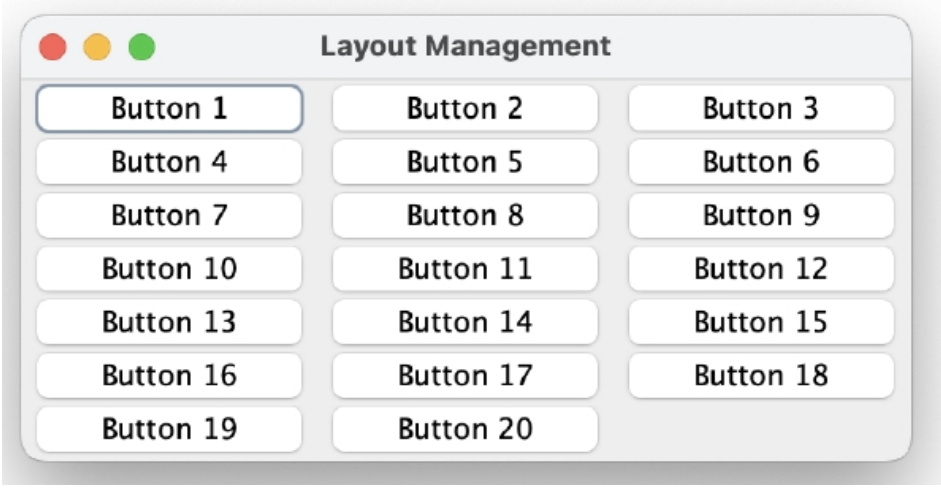
FlowLayout

If we add the following line of the code to previous example, and remove BorderLayout setting

```
setLayout(new FlowLayout());
```

GridLayout

```
public class LayoutFrame extends JFrame {
    public LayoutFrame(String title){
        super(title);
        this.setBounds(100, 100, 400, 200);
        setLayout(new GridLayout(7, 3)); // GridLayout

        for(int i = 0; i < 20; i++)
            add(new JButton("Button " + i));

        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {...}
}
```



GridBagLayout

```
public class LayoutDemo {
    public LayoutDemo() {
        JFrame frame = new JFrame("GridBagLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);

        JPanel panel = new JPanel(new GridBagLayout());
        frame.add(panel);

        // Create components
        JLabel nameLabel = new JLabel("Name:");
        JTextField nameField = new JTextField(5);
        JButton submitButton = new JButton("Submit");
        JButton returnButton = new JButton("Return");

        // Create GridBagConstraints
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);

        // Adding components to the panel
        panel.add(nameLabel, gbc);
        panel.add(nameField, gbc);
        panel.add(submitButton, gbc);
        panel.add(returnButton, gbc);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new LayoutDemo();
    }
```

BoxLayout

```
public class LayoutFrame extends JFrame {
    public LayoutFrame(String title) {
        super(title);
        this.setBounds(100, 100, 400, 200);
        JPanel verticalPanel = new JPanel();
        verticalPanel.setLayout(new BoxLayout(verticalPanel, BoxLayout.Y_AXIS));
        verticalPanel.add(new JButton("Button 1"));
        verticalPanel.add(new JButton("Button 2"));
        verticalPanel.add(new JButton("Button 3"));
        JPanel mainPanel = new JPanel();
        mainPanel.add(verticalPanel);
        add(mainPanel);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new LayoutFrame("Layout Management");
    }
}
```

Inner Class

```
public ListenerTypes(String title) {
    JButton jb1 = new JButton("Button");
    ActionListener bl = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JLabel jl1 = new JLabel();
            jl1.setText("Button is clicked.");
            add(jl1);
        }
    };
    jb1.addActionListener(bl);
    add(jb1);
}
```

Anonymous inner class example

```
JLabel jl1 = new JLabel();
JButton jb1 = new JButton("Button");
jb1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jl1.setText("Button is clicked.");
    }
});
add(jl1); add(jb1);
```

All event and listener types

User Action	Event Type Generated	Source Object
Click a button	ActionEvent	JButton
Mouse button pressed, released, etc...	MouseEvent	Component
Key pressed, released, etc...	KeyEvent	Component
Window opened, closed, etc...	WindowEvent	Window
Press return on a text field	ItemEvent, Action Event	JTextField
Click a check box	ItemEvent, ActionEvent	JCheckBox
Click a radio button	ItemEvent, ActionEvent	JRadioButton
Selecting a new item from a combo box	ItemEvent, ActionEvent	JComboBox

Event and listener types

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseClicked(MouseEvent) mouseExited(MouseEvent) mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTypeed(KeyEvent)

Example on MouseListener

```
public ListenerTypes(String title) {
    super(title);
    this.setBounds(100, 100, 400, 200);
    this.setLayout(new FlowLayout());
    JLabel jl1 = new JLabel();
    JButton jb1 = new JButton("Button");
    add(jl1);
    add(jb1);

    jb1.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            jl1.setText("Button is clicked.");
        }

        @Override
        public void mouseEntered(MouseEvent e) {
            jl1.setText("Mouse entered the button!");
        }
    });
```

Using Multiple Listeners in Java

```
public class ListenerTypes extends JFrame {
    public ListenerTypes(String title) {
        super(title);
        this.setBounds(100, 100, 400, 200);
        this.setLayout(new GridLayout(3, 1));

        JLabel jl1 = new JLabel();
        JButton jb1 = new JButton("Button1");
        JButton jb2 = new JButton("Button2");
        this.add(jb1);
        this.add(jl1);
        this.add(jb2);

        jb1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jl1.setText("Button 1 is clicked.");
            }
        });

        jb2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jl1.setText("Button 2 is clicked.");
            }
        });
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new ListenerTypes("Multiple Listeners-Part I");
    }
}
```

Using Multiple Listeners with if-else

```
public class ListenerTypes extends JFrame {
    JLabel jl1 = new JLabel();

    public ListenerTypes(String title) {
        super(title);
        JButton jb1 = new JButton("Button 1");
        JButton jb2 = new JButton("Button 2");

        jb1.addActionListener(new myactionL());
        jb2.addActionListener(new myactionL());

        add(jb1);
        add(jl1);
        add(jb2);

        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private class myactionL implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if (command.equals("Button 2")) {
                jl1.setText("Button 2 is clicked.");
            } else if (command.equals("Button 1")) {
                jl1.setText("Button 1 is clicked.");
            }
        }
    }
```