GUI programming in Java

Yulia Newton, Ph.D.

CS151, Object Oriented Programming

San Jose State University

Spring 2020

GUI programming in Java

- Many programming libraries are available
- Natural framework for object oriented programming
  - GUI components are objects
  - Abstraction, reuse, encapsulation, polymorphism, composition
- No need to re-invent the wheel!
  - Use libraries
  - Inherit from library provided classes
  - Writing your own GUI programming framework is more work than most programs require

Java libraries available for GUI programming

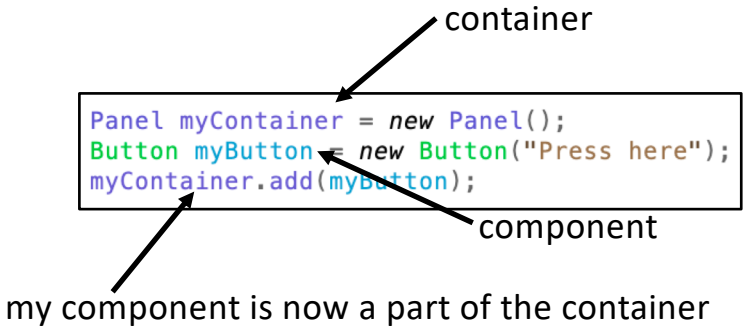
- AWT API
  - Introduced in Java 1 and by now has been mostly replace by other libraries and frameworks
- Java Swing
  - Set of graphic libraries
  - Most commonly currently used
- JavaFX
  - Introduced in Java 8 and is meant to replace Java Swing

Short introduction to Java AWT

- Provided by Java API
- Includes 12 packages
- Platform independent, device independent
- Resource-heavy
- *java.awt*
  - This package contains classes that represent UI objects
    - E.g. *Button*, *TextField*, *Label*, *Frame*
  - Layout managers
    - E.g. *FlowLayout*, *BorderLayout*, *GridLayout*
  - Graphics classes
    - E.g. *Graphics*, *Color, Font*
- *java.awt.event*
  - Event handling package
  - Event classes
    - E.g. *ActionEvent*, *MouseEvent*, *KeyEvent*
  - Listener interfaces
    - E.g. *ActionListener*, *MouseListener*, *MouseMotionListener*, *KeyListener*
  - Adapter classes
    - E.g. *MouseAdapter*, *KeyAdapter*
- Objects ”listen” for events to occur and handle these events according to the implementation

Containers vs. components

- Components
  - E.g. *Button*, *TextField*, *Label*
- Containers
  - E.g. *Frame*, *Panel*
- Containers contain components
- Inherit from standard container classes



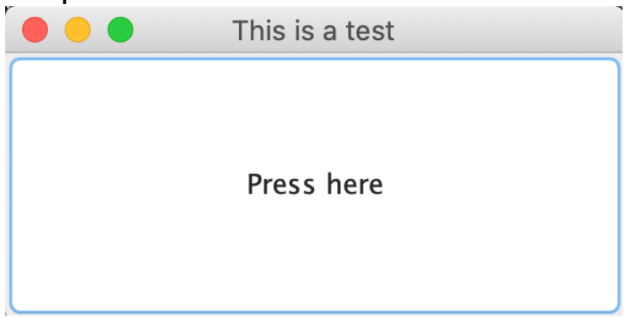
Example: basic popup with a button (no function)

```java
import java.awt.*;
import java.awt.event.*;

public class Test {
    public static void main(String[]args) {
        Frame myFrame = new Frame( "This is a test" );
        Button myButton = new Button("Press here");
        myFrame.add(myButton);
        myFrame.show();
    }
}
```

```java
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame {
    Test(){super();}
    Test(String title){super(title);}

    public static void main(String[]args) {
        Test myFrame = new Test( "This is a test" );
        Button myButton = new Button("Press here");
        myFrame.add(myButton);
        myFrame.show();
    }
}
```

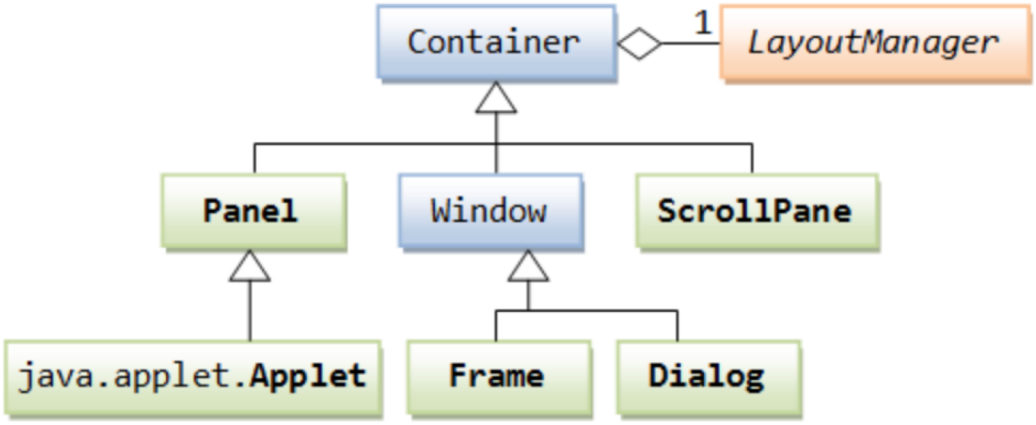
Different implementations with the same result

Output:



AWT containers

- *Frame* – popup window
- *Panel* – rectangular box in a window
- *ScrollPanel* – similar to *Panel* but automatically provides scroll bars



https://www.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

AWT components

- Various objects in a container that listen for user-initiated events and act upon those events according to implementation

https://www.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

AWT components API

Component	Constructors	Useful methods

Label	public Label(String label, int alignment);
	public Label(String label);
	public Label();
		public String getText();
		public void setText(String *label*);
		public int getAlignment();
		public void setAlignment(int *alignment*); // Label.LEFT, Label.RIGHT, Label.CENTER

TextField	public TextField(String text, int columns);
	public TextField(String text);
	public TextField(int columns);
		public String getText();
		public void setText(String text);
		public void setEditable(boolean editable);

Button	public Button(String label);
	public Button();
		public String getLabel();
		public void setLabel(String label);
		public void setEnabled(boolean enable);

Choice	Choice()	public void add(String item);
		public String getItem(int index);
		public int getItemCount();
		public void remove(int position);
		public void remove(String item);
		public void select(int pos);
		public void select(String str);
		public String getSelectedItem();
		public int getSelectedIndex();
		public void setEnabled(boolean enable);

Note: these are just some of the available public methods. See API for more information.

AWT components API (cont'd)

Component	Constructors	Useful methods

List	List()
	List(int rows)
	List(int rows, boolean multipleMode)
		public void add(String item);
		public void add(String item, int index);
		public void deselect(int index);
		public String getItem(int index);
		public int getItemCount();
		public String[] getItems();
		public String getSelectedItem();
		public String[] getSelectedItems();
		public boolean isMultipleMode();
		public void remove(int position);
		public void remove(String item);
		public void select(int index);
		public void setEnabled(boolean enable);

Checkbox	Checkbox()
	Checkbox(String label)
	Checkbox(String label, boolean state)
	Checkbox(String label, boolean state, CheckboxGroup group)
	Checkbox(String label, CheckboxGroup group, boolean state)
		public CheckboxGroup getCheckboxGroup();
		public String getLabel();
		public boolean getState();
		public void setLabel(String label);
		public void setState(boolean state);
		public void setEnabled(boolean enable);

Note: these are just some of the available public methods. See API for more information.

Anonymous vs. named instance

- Anonymous instances cannot be referenced in the code after instantiation

Label:

```
myContainer.add(new Label("This is a test label", Label.RIGHT));

Label myLabel = new Label("This is a test label", Label.RIGHT));
myContainer.add(myLabel);
```

anonymous instance

named instance

TexField:

```
myContainer.add(new TextField("This is a text field"));

TextField myTextfield = new TextField("This is a text field"));
myContainer.add(myTextfield);
```

anonymous instance

named instance

Button:

```
myContainer.add(new Button("This is a button"));

Button myButton = new Button("This is a button"));
myContainer.add(myButton);
```

anonymous instance

named instance

AWT event handling

- *java.awt.event* package
- Source object – object for which an event is initiated by the user
- Event class – represents a particular action, tied to a source object
- Listener object – listens for events initiated by the user
  - *myComponent.addActionListener(this);*
  - Can be anonymous instantiation
- Source object is triggered -> creates an event object -> publishes to all listener objects registered to the source object

Source – listener contract

- Let's say a source is capable of firing an event of a particular type
  - E.g. *MouseEvent*
- The listener implements a particular listener interface
  - E.g. *MouseListener*
  - Handler methods to be implemented
- Add the listener to the source object that will trigger the event to be listened for
  - *addMouseListener(MouseListener myListener);*

AWT built-in listeners

Interface	Description
ActionListener	The listener interface for receiving action events.
AdjustmentListener	The listener interface for receiving adjustment events.
AWTEventListener	The listener interface for receiving notification of events dispatched to objects that are instances of Component or MenuComponent or their subclasses.
ComponentListener	The listener interface for receiving component events.
ContainerListener	The listener interface for receiving container events.
FocusListener	The listener interface for receiving keyboard focus events on a component.
HierarchyBoundsListener	The listener interface for receiving ancestor moved and resized events.
HierarchyListener	The listener interface for receiving hierarchy changed events.
InputMethodListener	The listener interface for receiving input method events.
ItemListener	The listener interface for receiving item events.
KeyListener	The listener interface for receiving keyboard events (keystrokes).
MouseListener	The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.
MouseMotionListener	The listener interface for receiving mouse motion events on a component.
MouseWheelListener	The listener interface for receiving mouse wheel events on a component.
TextListener	The listener interface for receiving text events.
WindowFocusListener	The listener interface for receiving `WindowEvents`, including `WINDOW_GAINED_FOCUS` and `WINDOW_LOST_FOCUS` events.
WindowListener	The listener interface for receiving window events.
WindowStateListener	The listener interface for receiving window state events.

https://docs.oracle.com/

AWT MouseListener interface

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the `MouseMotionListener`.)

The class that is interested in processing a mouse event either implements this interface (and all the methods it contains) or extends the abstract `MouseAdapter` class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's `addMouseListener` method. A mouse event is generated when the mouse is pressed, released clicked (pressed and released). A mouse event is also generated when the mouse cursor enters or leaves a component. When a mouse event occurs, the relevant method in the listener object is invoked, and the `MouseEvent` is passed to it.

Modifier and Type	Method and Description
void	**mouseClicked**(**MouseEvent** e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(**MouseEvent** e) Invoked when the mouse enters a component.
void	**mouseExited**(**MouseEvent** e) Invoked when the mouse exits a component.
void	**mousePressed**(**MouseEvent** e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(**MouseEvent** e) Invoked when a mouse button has been released on a component.

https://docs.oracle.com/

AWT ActionListener interface

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.

Modifier and Type	Method and Description
`void`	**`actionPerformed`**`(`**`ActionEvent`**` e)` Invoked when an action occurs.

https://docs.oracle.com/

Example: count incrementing GUI

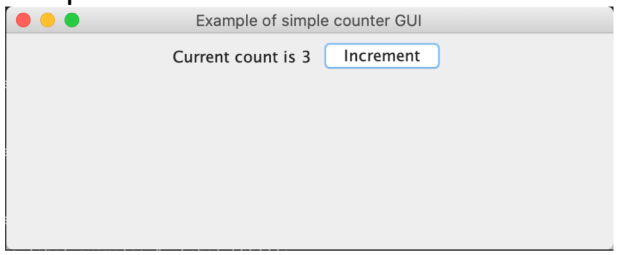
```
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame implements ActionListener {
    private Label countLabel;
    private Button countButton;
    private int count;

    public Test() {
        setLayout(new FlowLayout());

        count = 1;

        countLabel = new Label("Current count is "+count);
        add(countLabel);

        countButton = new Button("Increment");
        add(countButton);
        countButton.addActionListener(this);

        setTitle("Example of simple counter GUI");
        setSize(500, 200);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        count++;
        countLabel.setText("Current count is "+count);
    }

    public static void main(String[]args) {
        Test myApplication = new Test();
    }
}
```

Our class implements *ActionListener* interface

Setup our GUI components in the constructor

We can use FlowLayout for component layout

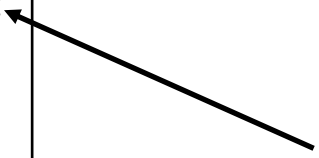
Output:



We implement *actionPerformed()* method from *ActionListener* interface. This is our event handler for Test class.

Example: the same incrementor app but with GUI app in a separate class

```
import java.awt.*;
import java.awt.event.*;

class myGUIApp extends Frame implements ActionListener {
    private Label countLabel;
    private Button countButton;
    private int count;

    public myGUIApp() {
        setLayout(new FlowLayout());

        count = 1;

        countLabel = new Label("Current count is "+count);
        add(countLabel);

        countButton = new Button("Increment");
        add(countButton);
        countButton.addActionListener(this);

        setTitle("Example of simple counter GUI");
        setSize(500, 200);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        count++;
        countLabel.setText("Current count is "+count);
    }
}

public class Test {
    public static void main(String[]args) {
        myGUIApp myApplication = new myGUIApp();
    }
}
```



Class myGUIApp is now responsible for all the GUI responsibilities and the main class simply creates myGUIApp instance

Example: adding window level event handling

```
import java.awt.*;
import java.awt.event.*;

class myGUIApp extends Frame implements ActionListener, WindowListener {
    private Label countLabel;
    private Button countButton;
    private int count;

    public myGUIApp() {
        setLayout(new FlowLayout());

        count = 1;

        countLabel = new Label("Current count is "+count);
        add(countLabel);

        countButton = new Button("Increment");
        add(countButton);
        countButton.addActionListener(this);

        addWindowListener(this);

        setTitle("Example of simple counter GUI");
        setSize(500, 200);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        count++;
        countLabel.setText("Current count is "+count);
    }

    @Override
    public void windowClosing(WindowEvent event) {
        System.exit(0);
    }
    @Override public void windowOpened(WindowEvent event) { }
    @Override public void windowClosed(WindowEvent event) { }
    @Override public void windowIconified(WindowEvent evt) { }
    @Override public void windowDeiconified(WindowEvent evt) { }
    @Override public void windowActivated(WindowEvent evt) { }
    @Override public void windowDeactivated(WindowEvent evt) { }
}

public class Test {
    public static void main(String[]args) {
        myGUIApp myApplication = new myGUIApp();
    }
}
```

Listen for window-level events

Add *WindowListener* to this class

Because we implement *WindowListener* interface we must provide implementation for all its methods, even if an empty implementation

Example: listening for mouse events

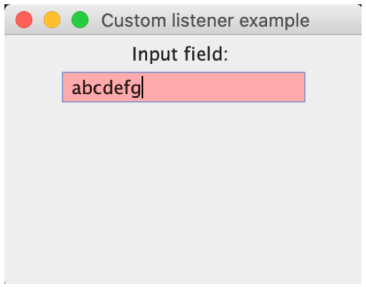
```java
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame implements MouseListener, MouseMotionListener {
    private Label xLabel;
    private Label yLabel;

    public Test() {
        setLayout(new FlowLayout());

        xLabel = new Label("x position: xxxx");
        add(xLabel);

        yLabel = new Label("y position: xxxx");
        add(yLabel);

        addMouseListener(this);

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

    @Override
    public void mouseClicked(MouseEvent event) {
        xLabel.setText("x position: "+event.getX());
        yLabel.setText("y position: "+event.getY());
    }

    @Override public void mousePressed(MouseEvent event) { }
    @Override public void mouseReleased(MouseEvent event) { }
    @Override public void mouseEntered(MouseEvent event) { }
    @Override public void mouseExited(MouseEvent event) { }
    @Override public void mouseDragged(MouseEvent event) { }
    @Override public void mouseMoved(MouseEvent event) { }

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

Don’t forget to add the listener to the frame (this class)

Even if these methods have an empty implementation we still need to include them because our class implements *MouseListener* and *MouseMotionListener* interfaces

Example: implementing key listeners

Output:



TextField is listening for keyboard events

Create a list of colors

Generate a random index in the list of colors and change the text field background to that color

```java
import java.awt.*;
import java.awt.Color;
import java.awt.event.*;
import java.util.*;
import java.util.Random;

public class Test extends Frame implements KeyListener {
    private Label myLabel;
    private TextField inputField;
    private ArrayList<Color> bckgrndColors;

    public Test() {
        setLayout(new FlowLayout());

        myLabel = new Label("Input field: ");
        add(myLabel);

        inputField = new TextField(20);
        add(inputField);
        inputField.addKeyListener(this);

        bckgrndColors = new ArrayList<Color>();
        bckgrndColors.add(Color.LIGHT_GRAY);
        bckgrndColors.add(Color.MAGENTA);
        bckgrndColors.add(Color.WHITE);
        bckgrndColors.add(Color.ORANGE);
        bckgrndColors.add(Color.PINK);
        bckgrndColors.add(Color.GREEN);
        bckgrndColors.add(Color.BLUE);
        bckgrndColors.add(Color.YELLOW);
        bckgrndColors.add(Color.CYAN);

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

    @Override
    public void keyTyped(KeyEvent event) {
        Random rand = new Random();
        int index = rand.nextInt(bckgrndColors.size());
        inputField.setBackground(bckgrndColors.get(index));
    }

    @Override public void keyPressed(KeyEvent event) { }
    @Override public void keyReleased(KeyEvent event) { }

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

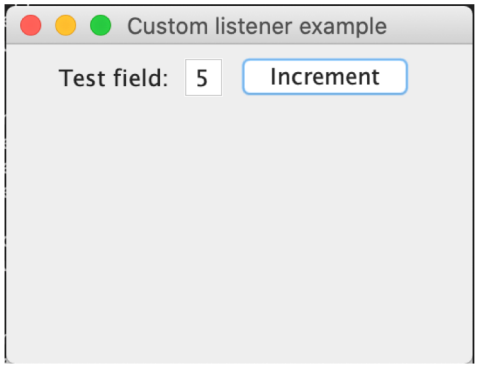
Example: implementing custom listener

```
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame {
    private Label myLabel;
    private TextField myField;
    private Button myButton;
    private Integer count;

    public Test() {
        setLayout(new FlowLayout());
        count = 1;

        myLabel = new Label("Test field: ");
        add(myLabel);

        myField = new TextField();
        add(myField);
        myField.setText(count.toString());

        myButton = new Button("Increment");
        add(myButton);
        myButton.addActionListener(new CustomListener());

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

    private class CustomListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent event) {
            count++;
            myField.setText(count.toString());
        }
    }

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

Don’t forget to add the listener to the object that should listen to events

Custom listener class that implements ActionListener

Output:

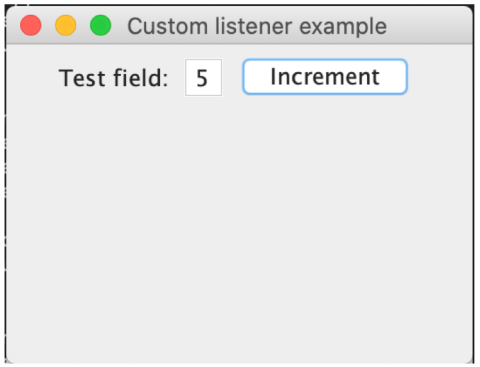


Example: anonymous listener inner class

```
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame {
    private Label myLabel;
    private TextField myField;
    private Button myButton;
    private Integer count;

    public Test() {
        setLayout(new FlowLayout());
        count = 1;

        myLabel = new Label("Test field: ");
        add(myLabel);

        myField = new TextField();
        add(myField);
        myField.setText(count.toString());

        myButton = new Button("Increment");
        add(myButton);
        myButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                count++;
                myField.setText(count.toString());
            }
        });

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

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

Add a listener as an anonymous inner class

Output:

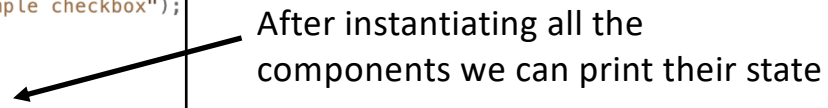


Example: using *toString()* with AWT objects

```
import java.awt.*;
import java.awt.event.*;

public class Test extends Frame {
    private Label myLabel;
    private TextField myField;
    private Button myButton;
    private Choice myChoice;
    private List myList;
    private Checkbox myCheckbox;

    public Test() {
        myLabel = new Label("Sample label");

        myField = new TextField("Sample text",20);

        myButton = new Button("Sample button");

        myChoice = new Choice();

        myList = new List();
        myList.add("Item1");
        myList.add("Item2");

        myCheckbox = new Checkbox("Sample checkbox");

        System.out.println(this);
        System.out.println(myLabel);
        System.out.println(myField);
        System.out.println(myButton);
        System.out.println(myChoice);
        System.out.println(myList);
        System.out.println(myCheckbox);
    }

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

Output:

```
Test[frame0,0,23,0x0,invalid,hidden,layout=java.awt.BorderLayout,title=,resizable,normal]
java.awt.Label[label0,0,0,0x0,invalid,align=left,text=Sample label]
java.awt.TextField[textfield0,0,0,0x0,invalid,text=Sample text,editable,selection=0-0]
java.awt.Button[button0,0,0,0x0,invalid,label=Sample button]
java.awt.Choice[choice0,0,0,0x0,invalid,current=null]
java.awt.List[list0,0,0,0x0,invalid,selected=null]
java.awt.Checkbox[checkbox0,0,0,0x0,invalid,label=Sample checkbox,state=false]
```



Adapter classes

- Adapter classes implement listener interfaces
  - E.g. *WindowAdapter* implements *WindowListener*
- Allow us to only implement those listener events that we are interested in handling

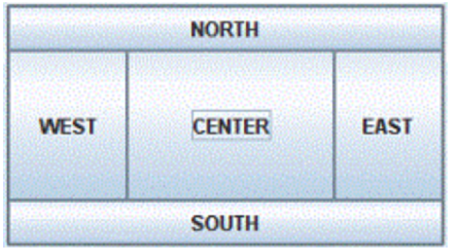
Example: WindowListener vs. WindowAdapter

```
public class Test extends Frame {
    private Label myLabel;
    private TextField myField;
    private Button myButton;
    private Integer count;

    public Test() {
        setLayout(new FlowLayout());
        count = 1;

        myLabel = new Label("Test field: ");
        add(myLabel);

        myField = new TextField();
        add(myField);
        myField.setText(count.toString());

        myButton = new Button("Increment");
        add(myButton);
        myButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                count++;
                myField.setText(count.toString());
            }
        });

        addWindowListener(new WindowListener() {
            @Override
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
            @Override public void windowOpened(WindowEvent event) { }
            @Override public void windowClosed(WindowEvent event) { }
            @Override public void windowIconified(WindowEvent event) { }
            @Override public void windowDeiconified(WindowEvent event) { }
            @Override public void windowActivated(WindowEvent event) { }
            @Override public void windowDeactivated(WindowEvent event) { }
        });

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

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

```
public class Test extends Frame {
    private Label myLabel;
    private TextField myField;
    private Button myButton;
    private Integer count;

    public Test() {
        setLayout(new FlowLayout());
        count = 1;

        myLabel = new Label("Test field: ");
        add(myLabel);

        myField = new TextField();
        add(myField);
        myField.setText(count.toString());

        myButton = new Button("Increment");
        add(myButton);
        myButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                count++;
                myField.setText(count.toString());
            }
        });

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        });

        setTitle("Custom listener example");
        setSize(200, 200);
        setVisible(true);
    }

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

GUI AWT layouts

- *java.awt.FlowLayout* – components are arranged left to right in the order they are added
- *java.awt.GridLayout* – components are arranged in the grid of rows and columns (matrix)
- *java.awt.BoxLayout* – components are arranged in one row or one column, independent of the size of the container
- *java.awt.BorderLayout* – components are arranged in the container among 5 zones (NORTH, WEST, CENTER, EAST, SOUTH)



https://www.ntu.edu.sg/

Example: calculator app

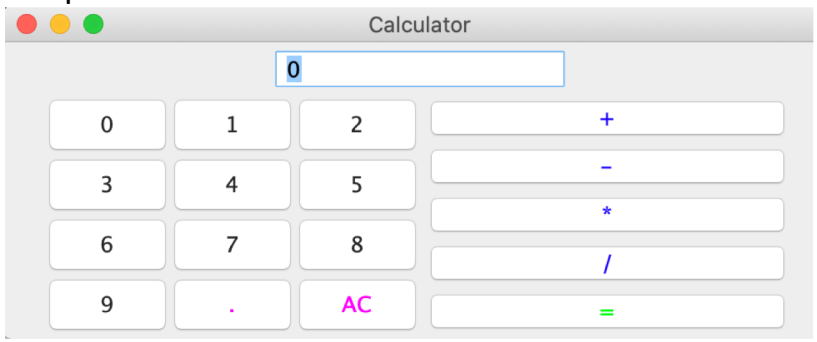
```
import java.awt.*;
import java.awt.Color;
import java.awt.event.*;
import java.util.*;
```

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...
    }


    private class OperatorListener implements ActionListener {
        ...
    }

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

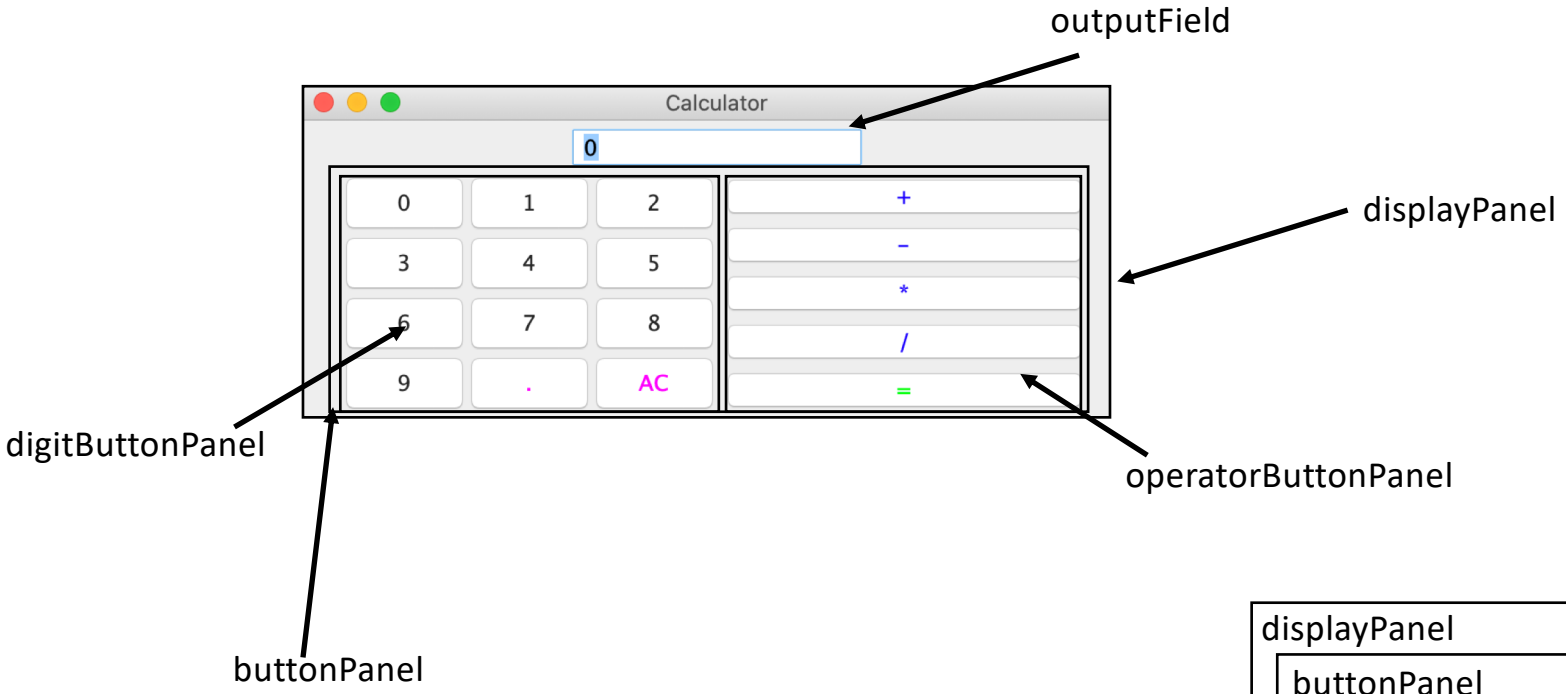
Class fields that make up the GUI and functionality

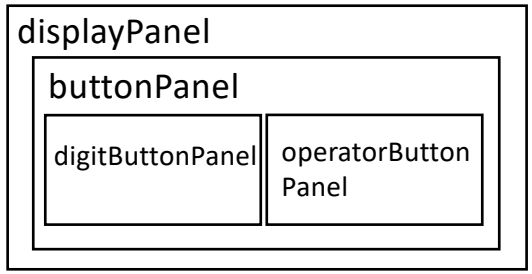
Run the program

Output:



Example: calculator app (cont'd)

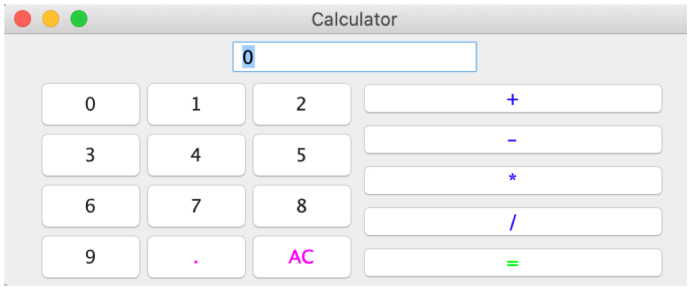


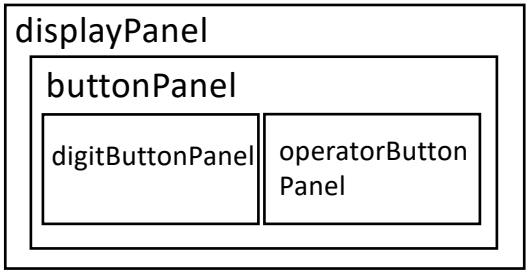


Example: calculator app (cont'd)

constructor

```
public Test() {
    this.currentOperation = "";
    this.firstOperand = 0.0;

    Panel displayPanel = new Panel(new FlowLayout());
    outputField = new TextField("0", 20);
    displayPanel.add(outputField);

    Panel buttonPanel = new Panel(new GridLayout(1, 2));

                    ...
```



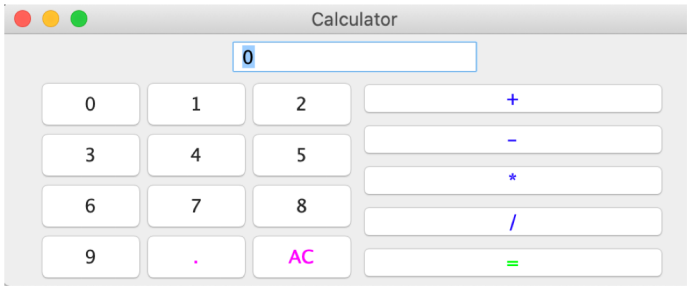


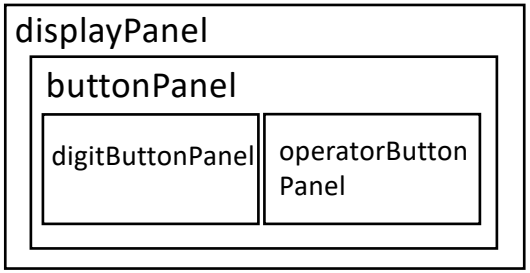
Example: calculator app (cont'd)

...

```
Panel digitButtonPanel = new Panel(new GridLayout(4, 3));
digits = new ArrayList<Button>();
digits.add(new Button("0"));
digits.add(new Button("1"));
digits.add(new Button("2"));
digits.add(new Button("3"));
digits.add(new Button("4"));
digits.add(new Button("5"));
digits.add(new Button("6"));
digits.add(new Button("7"));
digits.add(new Button("8"));
digits.add(new Button("9"));
digits.add(new Button("."));
digits.add(new Button("AC"));

digits.get(10).setForeground(Color.MAGENTA);
digits.get(11).setForeground(Color.MAGENTA);

digitButtonPanel.add(digits.get(0));
digitButtonPanel.add(digits.get(1));
digitButtonPanel.add(digits.get(2));
digitButtonPanel.add(digits.get(3));
digitButtonPanel.add(digits.get(4));
digitButtonPanel.add(digits.get(5));
digitButtonPanel.add(digits.get(6));
digitButtonPanel.add(digits.get(7));
digitButtonPanel.add(digits.get(8));
digitButtonPanel.add(digits.get(9));
digitButtonPanel.add(digits.get(10));
digitButtonPanel.add(digits.get(11));
buttonPanel.add(digitButtonPanel);
```

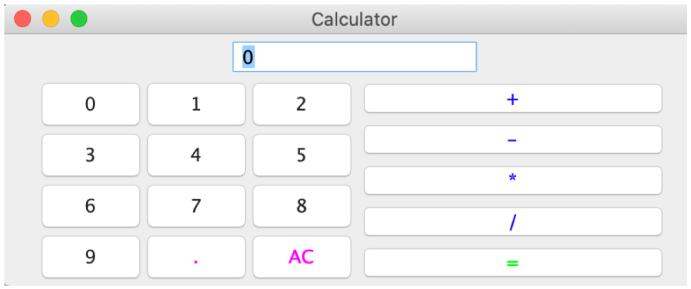
...

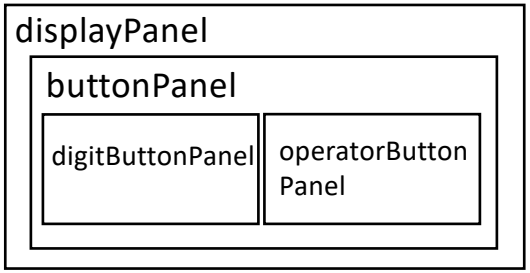




Example: calculator app (cont'd)

```
...
Panel operatorButtonPanel = new Panel(new GridLayout(5, 1));
plusSign = new Button("+");
minusSign = new Button("-");
multSign = new Button("*");
divSign = new Button("/");
eqSign = new Button("=");

plusSign.setForeground(Color.BLUE);
minusSign.setForeground(Color.BLUE);
multSign.setForeground(Color.BLUE);
divSign.setForeground(Color.BLUE);
eqSign.setForeground(Color.GREEN);

operatorButtonPanel.add(plusSign);
operatorButtonPanel.add(minusSign);
operatorButtonPanel.add(multSign);
operatorButtonPanel.add(divSign);
operatorButtonPanel.add(eqSign);
buttonPanel.add(operatorButtonPanel);

displayPanel.add(buttonPanel);
add(displayPanel);

...
```





Example: calculator app (cont'd)

```
...
digits.get(11).addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        //currentOperation = "";
        //firstOperand = 0.0;
        //outputField.setText("0");
        resetValues();
    }
});

digits.get(10).addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        String currentText = outputField.getText();
        if(currentText.indexOf(".") < 0){
            outputField.setText(currentText+".");
        }
    }
});

...
```

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...


    private class OperatorListener implements ActionListener {
        ...
    }

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

```
private void resetValues(){
    currentOperation = "";
    firstOperand = 0.0;
    outputField.setText("0");
    outputField.setBackground(Color.WHITE);
}
```

Example: calculator app (cont'd)

```
...
OperatorListener opListener = new OperatorListener();
plusSign.addActionListener(opListener);
minusSign.addActionListener(opListener);
multSign.addActionListener(opListener);
divSign.addActionListener(opListener);
...
```

Inside the constructor

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...
    }


    private class OperatorListener implements ActionListener {
        ...
    }

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

```
private class OperatorListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        Button source = (Button)event.getSource();
        if (source == plusSign) {
            currentOperation = "+";
        }else if (source == minusSign) {
            currentOperation = "-";
        }else if (source == multSign) {
            currentOperation = "*";
        }else if (source == divSign) {
            currentOperation = "/";
        }

        String currentText = outputField.getText();
        try{
            Double currentTextDouble = new Double(currentText);
            firstOperand = currentTextDouble;
            outputField.setText("0");
        } catch(NumberFormatException e){
            resetValues();
        }
    }
}
```

Example: calculator app (cont'd)

```
...
for(int i = 0; i <= 9; i ++){
    digits.get(i).addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent event) {
            String currentText = outputField.getText();
            Button source = (Button)event.getSource();
            String newDigit = "";
            if (source == digits.get(0)) {
                newDigit = "0";
            } else if (source == digits.get(1)) {
                newDigit = "1";
            } else if (source == digits.get(2)) {
                newDigit = "2";
            } else if (source == digits.get(3)) {
                newDigit = "3";
            } else if (source == digits.get(4)) {
                newDigit = "4";
            } else if (source == digits.get(5)) {
                newDigit = "5";
            } else if (source == digits.get(6)) {
                newDigit = "6";
            } else if (source == digits.get(7)) {
                newDigit = "7";
            } else if (source == digits.get(8)) {
                newDigit = "8";
            } else if (source == digits.get(9)) {
                newDigit = "9";
            }

            currentText = currentText + newDigit;
            currentText = currentText.replaceFirst("^0+(?!$)", "");
            outputField.setText(currentText);
        }
    });
}
...
```

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...
    }


    private class OperatorListener implements ActionListener {
        ...
    }

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

Example: calculator app (cont'd)

```
...
eqSign.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        Double result = 0.0;
        String currentText = outputField.getText();
        try{
            Double secondOperand = new Double(currentText);

            if(currentOperation == "+"){
                result = firstOperand + secondOperand;
            } else if(currentOperation == "-"){
                result = firstOperand - secondOperand;
            } else if(currentOperation == "*"){
                result = firstOperand * secondOperand;
            } else if(currentOperation == "/"){
                if(secondOperand != 0.0){
                    result = firstOperand / secondOperand;
                } else {
                    resetValues();
                    outputField.setBackground(Color.PINK);
                }
            }

            outputField.setText(result.toString());
            firstOperand = result;
        } catch(NumberFormatException e){
            resetValues();
        }
    }
});
...
```

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...
    }


    private class OperatorListener implements ActionListener {
        ...
    }

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

Example: calculator app (cont'd)

```
                    ...
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent event) {
        System.exit(0);
    }
});

setTitle("Calculator");
setSize(500, 200);
setVisible(true);

                    ...
```

```
public class Test extends Frame {
    private Label myLabel;
    private TextField outputField;
    private ArrayList<Button> digits;
    private Button plusSign;
    private Button minusSign;
    private Button multSign;
    private Button divSign;
    private Button eqSign;
    private String currentOperation;
    private Double firstOperand;

    public Test() {
        ...
    }

    private void resetValues(){
        ...
    }


    private class OperatorListener implements ActionListener {
        ...
    }

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

Java Swing

- Java Swing is a lightweight Java GUI programming toolkit
  - Resource-light
- Allows high quality 2-D graphics and images
- 18 packages, >700 classes
- Java Swing classes begin with “J”
  - E.g. *Jbutton*
  - *javax.swing* package
- Java Swing components
  - E.g. *JButton*, *JTextField*, *JLabel*, etc.
- Java Swing containers
  - E.g. *JFrame*, *JDialog*, *JPanel*, etc.

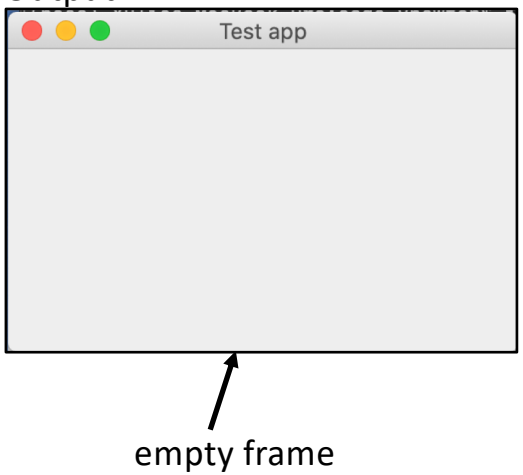
Java Swing containers

- *javax.swing.JFrame* – stand alone application frame
- *javax.swing.JApplet* – for running Java GUI code in a web page
- *javax.swing.JPanel* – allows laying out components in a frame
- *javax.swing.JScrollPane* – same but with scroll bars
- *javax.swing.JSplitPane* – allows displaying two panels separated by vertical or horizontal scroll bar
- *javax.swing.JTabbedPane* – allows displaying panels as tabs

Example: simple Java Swing app

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

public class Test extends JFrame {
    public Test(){
        setTitle("Test app");
        setSize(300,200);
        setLocation(10,200);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Output:



Example: adding program closing behavior

Using window listener event:

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

public class Test extends JFrame {
    public Test(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );

        setTitle("Test app");
        setSize(350,200);
        setLocation(10,200);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Using *JFrame* method:

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

public class Test extends JFrame {
    public Test(){
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setTitle("Test app");
        setSize(350,200);
        setLocation(10,200);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Example: adding a window icon

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

public class Test extends JFrame {
    public Test(){
        ImageIcon app_icon = new ImageIcon("./my_app_icon.png");
        setIconImage(app_icon.getImage());

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setTitle("Test app");
        setSize(350,200);
        setLocation(10,200);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Note: MacOS does not support frame icons

Ways to add components to your GUI app

- Using *add()* method JFrame inherits from Java AWT Frame class
- Using content pane: *getContentPane()* method
  - Named content pane instance
    - *Container content = getContentPane();*
    - *content.add(…);*
  - Unnamed content pane reference
    - *getContentPane().add(...);*
  - Using JPanel
    - *JPanel content = new JPanel();*
    - *content.add(...);*
    - *setContentPane(content);*

Content pane

- Every JFrame comes with a pre-created content pane
- It's recommended to place JComponents into content-pane
- *getContentPane()* returns a container object
  - Internally Java is using *JPanel*
- GUI application is a hierarchy of JPanel containers
- It might be a good idea to treat top level container as a JPanel from the beginning (last option mentioned in the previous slide)

Example: adding a button using *add() JFrame* method

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

public class Test extends JFrame {
    public Test(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );

        setTitle("Test app");
        setSize(300,200);
        setLocation(10,200);

        JButton myButton = new JButton("Test button");
        add(myButton);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Create a button instance and add it to our JFrame

Output:

Example: adding a button using named content pane instance

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

public class Test extends JFrame {
    public Test(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );

        setTitle("Test app");
        setSize(300,200);
        setLocation(10,200);

        Container contentPane = getContentPane();
        JButton myButton = new JButton("Test button");
        contentPane.add(myButton);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Create a named content pane instance

Create a button instance and add it to the content pane instance

Output:

Example: adding a button using unnamed content pane reference

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

public class Test extends JFrame {
    public Test(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );

        setTitle("Test app");
        setSize(300,200);
        setLocation(10,200);

        JButton myButton = new JButton("Test button");
        getContentPane().add(myButton);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Output:

Create a button instance and add it to the content pane instance, for which we obtain an unnamed reference using *getContentPane()* method

Example: adding a button using *JPanel*

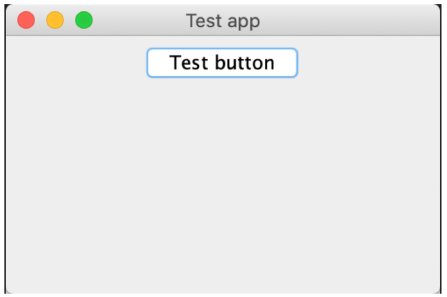
```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class Test extends JFrame {
    public Test(){
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.exit(0);
            }
        } );

        setTitle("Test app");
        setSize(300,200);
        setLocation(10,200);

        JPanel content = new JPanel();
        JButton myButton = new JButton("Test button");
        content.add(myButton);
        setContentPane(content);
    }

    public static void main(String[]args) {
        JFrame myApp = new Test();
        myApp.show();
    }
}
```

Create a new instance of JPanel

Create a button instance and add it to the instance of *JPanel*

Add the instance of *JPanel* to our application's content pane

Output:



Setting layout of the content pane

- Use the content pane's method setLayout()
  - *content.setLayout(new **XXX**Layout());*
    - Remember we can have named or unnamed instances of the content pane
- You can also set a layout of a *JPanel*

Java Swing layouts

- FlowLayout – arranges components in a directional flow in the order they are added
  - Can specify alignment and gaps in this layout's constructor
- BorderLayout – same as AWT BorderLayout, arranges components in 5 regions (NORTH, SOUTH, EAST, WEST and CENTER)
- CardLayout – treats components as a stack and you can only see a single component at a time
- BoxLayout – same as AWT BoxLayout, components are arranged in one row or one column, independent of the size of the container
- GridLayout – components are arranged in a grid/matrix format
- GridBagLayout – more flexible version of grid layout, allowing laying out components of unequal size
- SpringLayout – allows controlling vertical and horizontal distance between component edges
- GroupLayout – arranges components into hierarchy of sequential and/or parallel groups