OBJECT-ORIENTED PROGRAMMING

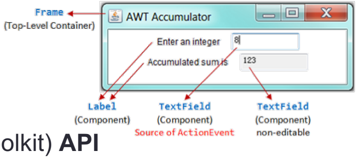
9. GUI PROGRAMMING WITH AWT & SWING

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

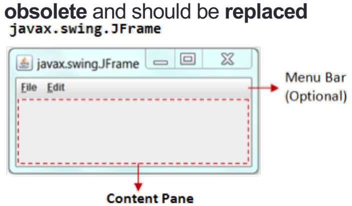
Outline

1. GUI Programming in Java
2. AWT UI Elements
3. AWT Event Handling
4. Swing GUI Elements
5. Layout Manager

AWT and Swing



- **AWT** (Abstract Windowing Toolkit) **API**
  - From JDK 1.0
  - Most components have **become obsolete** and should be **replaced by Swing** components



- **Swing** API
  - From JDK 1.1, as a part of **Java Foundation Classes** (JFC)
  - A much more comprehensive set of graphics libraries that enhances the AWT
  - JFC consists of **Swing**, *Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs.*

JavaFX

- **JavaFX** is a software platform for creating and delivering desktop applications, as well as **rich internet applications** (RIAs) that can run across a wide variety of devices.
- **JavaFX** is intended to replace **Swing** as the standard ***GUI library*** *for* ***Java SE***, but both will be included for the foreseeable future.

IFX is just a name, which is normally related with `sound or visual effects` *in the* ***javafx*** *i was in the belief that the* ***fx*** *was function. ...*
FIPS ***stands*** *for the Federal Information Processing Standardization*

Which should we choose?

- **AWT**: for simple GUI, but not for comprehensive ones
  - Native OS GUI
  - Flatform-independent and device-independent interface
  - Heavyweight components

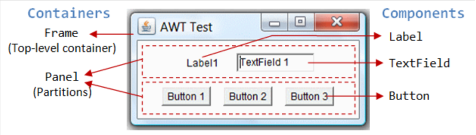
- **Swing**: Pure Java code with a more robust, versatile, and flexible library
  - Use AWT for windows and event handling
  - Pure-Java GUI, 100% portable and same across platform
  - Most components are light-weight, different look-and-feel
- **JavaFX:** for developing rich Internet applications
  - Can run across a wide variety of devices
  - More consistent in style and has additional options, e.g. 3D, chart, audio, video…

Outline

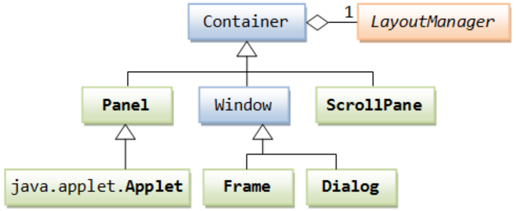
1. GUI Programming in Java
2. AWT UI Elements
3. AWT Event Handling
4. Swing GUI Elements
5. Layout Manager

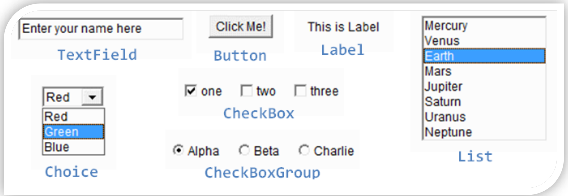
AWT UI Elements

- ***Component***: Components are elementary GUI entities, e.g. Button, Label, and TextField
  - **GUI components are also called *controls* (Microsoft ActiveX Control), *widgets* (Eclipse's Standard Widget Toolkit, Google Web Toolkit)**, which allow users to interact with the application through these components (*such as button-click and text-entry)*
- ***Container***: Containers (e.g. Frame, Panel and Applet) are used to *hold components in a specific layout* (such as flow or grid). A container can also hold sub-containers.

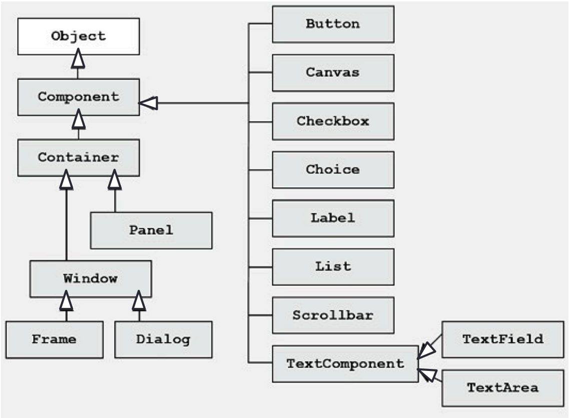


AWT Container Classes





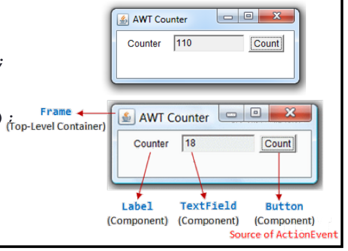
AWT Component Classes



Example: AWT Counter

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionListener;

public class AWTCounter
        extends Frame {
   private Label lblCount;
   private TextField tfCount;
   private Button btnCount;
   private int count = 0; // Counter's value

   // Setup GUI components
  public AWTCounter () {
      setLayout(new FlowLayout());

      lblCount = new Label("Counter");
      add(lblCount);

      tfCount = new TextField("0", 10);
      // set to read-only
      tfCount.setEditable(false);
      add(tfCount);
```

```
      btnCount =
           new Button("Count");
      add(btnCount);
      setTitle("AWT Counter");
      setSize(250, 100;

      setVisible(true);
   }
}
```



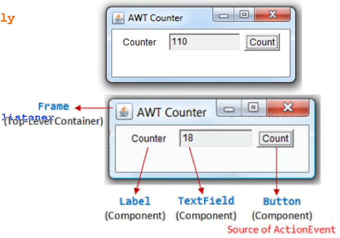
Outline

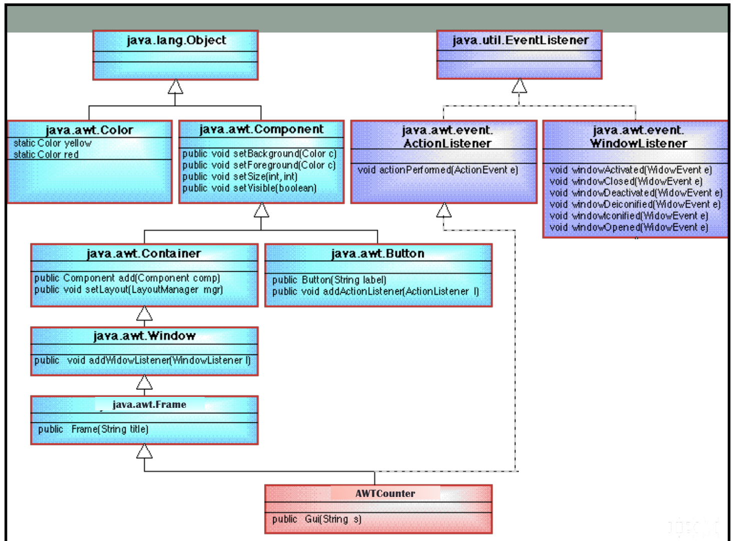
1. GUI Programming in Java
2. AWT UI Elements
3. AWT Event Handling
4. Swing GUI Elements
5. Layout Manager

Example: AWT Counter with Event Handling

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionListener;

public class AWTCounter
        extends Frame implements ActionListener {
   private Label lblCount;
   private TextField tfCount;
   private Button btnCount;
   private int count = 0; // Counter's value

   // Setup GUI components and event handling
  public AWTCounter () {
      setLayout(new FlowLayout());

      lblCount = new Label("Counter");
      add(lblCount);

      tfCount = new TextField("0", 10);
      tfCount.setEditable(false); // set to read-only
      add(tfCount);

      btnCount = new Button("Count");
      add(btnCount);

      //Clicking Button source fires ActionEvent
     //btnCount registers this instance as ActionEvent listener
      btnCount.addActionListener(this);

      setTitle("AWT Counter");
      setSize(250, 100;

      setVisible(true);
}
```

```
   public static void main(String[] args){
      AWTCounter app = new AWTCounter();
   }
    /** ActionEvent handler - Called back upon
button-click. */
   public void actionPerformed(ActionEvent e){
      ++count;
      // Display the counter value on the TextField
      tfCount.setText(count + "");
   }
}
```

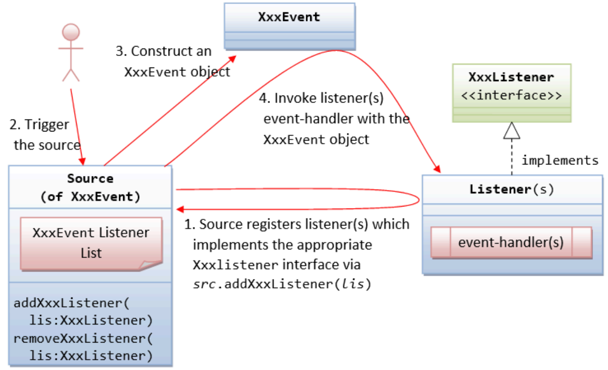




```
java.lang.Object

java.util.EventListener

java.awt.Color
static Color yellow
static Color red

java.awt.Component
public void setBackground(Color c)
public void setForeground(Color c)
public void setSize(int, int)
public void setVisible(boolean)

java.awt.event.
ActionListener
void actionPerformed(ActionEvent e)

java.awt.event.
WindowListener
void windowActivated(WidowEvent e)
void windowClosed(WidowEvent e)
void windowDeactivated(WidowEvent e)
void windowDeiconified(WidowEvent e)
void windowIconified(WidowEvent e)
void windowOpened(WidowEvent e)

java.awt.Container
public Component add(Component comp)
public void setLayout(LayoutManager mgr)

java.awt.Button
public Button(String label)
public void addActionListener(ActionListener l)

java.awt.Window
public  void addWidowListener(WindowListener l)

java.awt.Frame
public  Frame(String title)

AWTCounter
public  Gui(String s)
```

AWT Event Handling

Event Driven / Event Delegation



XxxEvent

3. Construct an XxxEvent object

2. Trigger the source

4. Invoke listener(s) event-handler with the XxxEvent object

XxxListener
<<interface>>

implements

Source
(of XxxEvent)

XxxEvent Listener List

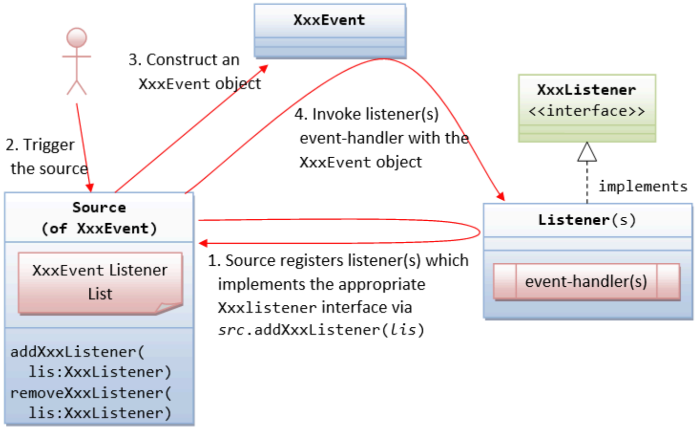
addXxxListener(
  lis:XxxListener)
removeXxxListener(
  lis:XxxListener)

1. Source registers listener(s) which implements the appropriate Xxxlistener interface via *src*.addXxxListener(*lis*)

Listener(s)

event-handler(s)

Quiz: Identify Source, Xxx for AWTCounter



XxxEvent

3. Construct an XxxEvent object

2. Trigger the source

4. Invoke listener(s) event-handler with the XxxEvent object

XxxListener
<<interface>>

implements

Source
(of XxxEvent)

XxxEvent Listener List

addXxxListener(
  lis:XxxListener)
removeXxxListener(
  lis:XxxListener)

1. Source registers listener(s) which implements the appropriate Xxxlistener interface via *src*.addXxxListener(*lis*)

Listener(s)

event-handler(s)

E.g. MouseListener (XxxListener) interface

```
//A MouseListener interface, which declares the signature of the handlers
//for the various operational modes.
public interface MouseListener {
   // Called back upon mouse-button pressed
   public void mousePressed(MouseEvent evt);
   // Called back upon mouse-button released
   public void mouseReleased(MouseEvent evt);
   // Called back upon mouse-button clicked (pressed and released)
   public void mouseClicked(MouseEvent evt);
   // Called back when mouse pointer entered the component
   public void mouseEntered(MouseEvent evt);
   // Called back when mouse pointer exited the component
   public void mouseExited(MouseEvent evt);
}
```

Add or remove XxxListener in the source:

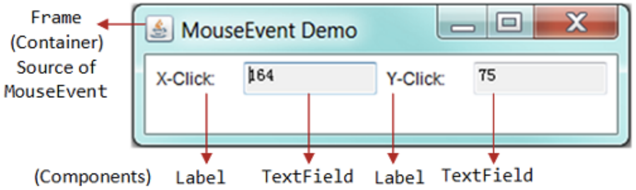
```
public void addXxxListener(XxxListener lis);
public void removeXxxListener(XxxListener lis);
```

```
//An example provides implementation to the event handler methods
class MyMouseListener implements MouseListener {
   @Override
   public void mousePressed(MouseEvent e)  {
      System.out.println("Mouse-button pressed!");
   }
   @Override
   public void mouseReleased(MouseEvent e) {
      System.out.println("Mouse-button released!");
   }
   @Override
   public void mouseClicked(MouseEvent e)  {
      System.out.println("Mouse-button clicked (pressed and released)!");
   }
   @Override
   public void mouseEntered(MouseEvent e)  {
      System.out.println("Mouse-pointer entered the source component!");
   }
   @Override
   public void mouseExited(MouseEvent e)   {
      System.out.println("Mouse exited-pointer the source component!");
   }
}
```

```
import java.awt.*;
import java.awt.event.*;
public class MouseEventDemo extends Frame {
  private TextField tfMouseX; // to display mouse-click-x
  private TextField tfMouseY; // to display mouse-click-y
  //setup the UI components and event handlers
  public MouseEventDemo() {
        setLayout(new FlowLayout());
        add(new Label("X-Click: "));
        tfMouseX = new TextField(10); // 10 columns
        tfMouseX.setEditable(false;     add(tfMouseX);
        add(new Label("Y-Click: "));    tfMouseY = new TextField(10);
        tfMouseY.setEditable(false);    add(tfMouseY);
         /* "super" frame (source) fires the MouseEvent and adds an anonymous
instance of MyMouseListener as a MouseEvent listener */
        addMouseListener(new MyMouseListener());
        setTitle("MouseEvent Demo");    setSize(350, 100);   setVisible(true);
  }
  public static void main(String[] args) {
        new MouseEventDemo();  // Let the constructor do the job
  }
}
```

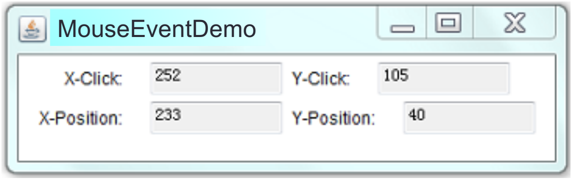
Quiz: MouseEventDemo

- Source: ?
- XxxEvent: ?
- What happens when running the program?



X-Click: 164 Y-Click: 75

Exercise: Modify MouseEventDemo with more mouse events



X-Click: 252 Y-Click: 105

X-Position: 233 Y-Position: 40

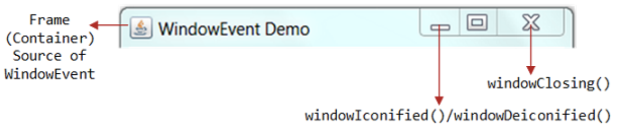
```
public interface MouseMotionListener {
  /* Called-back when a mouse-button is pressed on the
  source component and then dragged. */
  public void mouseDragged(MouseEvent e);
  /* Called-back when the mouse-pointer has been moved onto the
  source component but no buttons have been pushed. */
  public void mouseMoved(MouseEvent e);
}
```

Exercise: Modify AWTCounter with window events

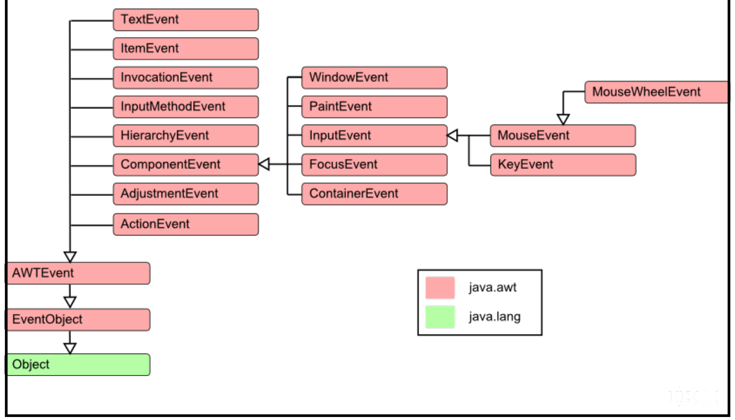
- Show the dialog box with a corresponding message for each event in WindowListener interface
  - close/open window
  - activate/deactivate window
  - iconify/deiconify window

java.awt.event.WindowListener

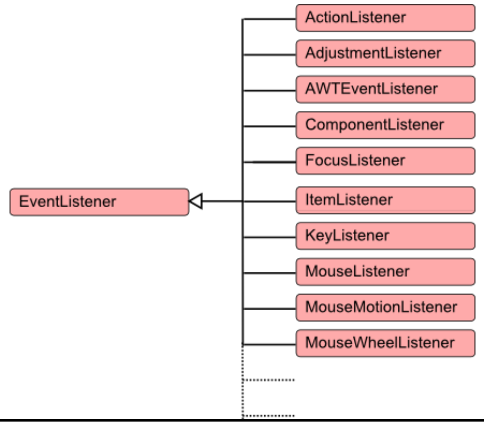
```
void windowActivated(WidowEvent e)
void windowClosed(WidowEvent e)
void windowDeactivated(WidowEvent e)
void windowDeiconified(WidowEvent e)
void windowIconified(WidowEvent e)
void windowOpened(WidowEvent e)
```



Event Hierarchy



EventListener Hierarchy



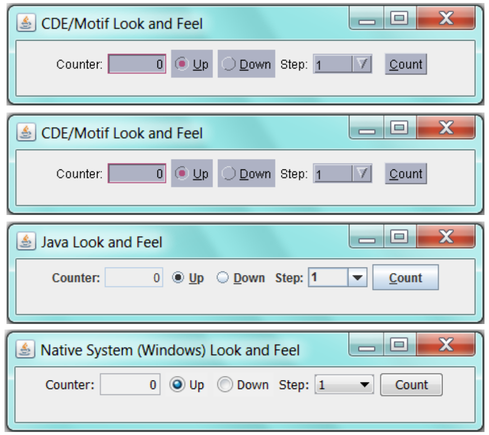
Outline

1. GUI Programming in Java
2. AWT UI Elements
3. AWT Event Handling
4. **Swing GUI Elements**
5. Layout Manager

Java Swing

- Light Weight: Pure Java code
  - Freelance of native operational System's API
- Use the Swing components with prefix "J", e.g. JFrame, JButton, JTextField, JLabel, etc.
  - Advanced controls like Tree, color picker, table controls, TabbedPane, slider.
- Uses the AWT event-handling classes
- Highly Customizable
  - Often made-to-order in a very simple method as visual appearance is freelance of content.
- Pluggable look-and-feel
  - Modified at run-time, supported by accessible values.

Swing – Different Look & Feel



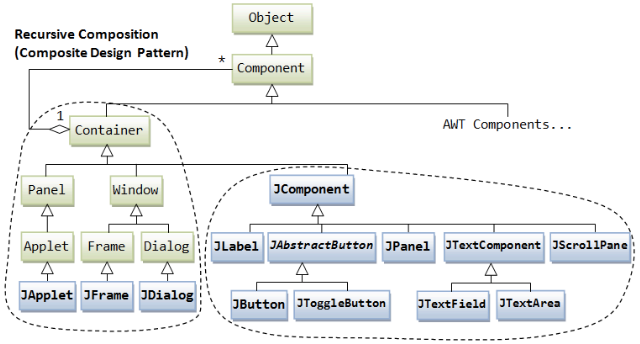
CDE/Motif Look and Feel
Counter: 0 ● Up ○ Down Step: 1 Count

CDE/Motif Look and Feel
Counter: 0 ● Up ○ Down Step: 1 Count

Java Look and Feel
Counter: 0 ● Up ○ Down Step: 1 Count

Native System (Windows) Look and Feel
Counter: 0 ● Up ○ Down Step: 1 Count

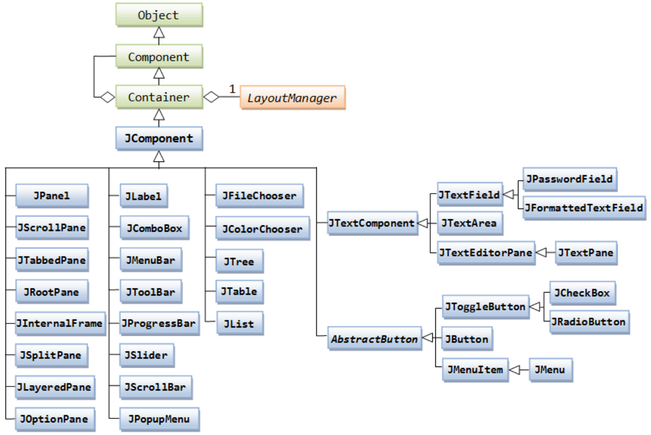
AWT and Swing Elements



Recursive Composition (Composite Design Pattern)

Object
Component
*
1
Container
AWT Components...
Panel
Window
Applet
Frame
Dialog
JApplet
JFrame
JDialog
JComponent
JLabel
JAbstractButton
JPanel
JTextComponent
JScrollPane
JButton
JToggleButton
JTextField
JTextArea

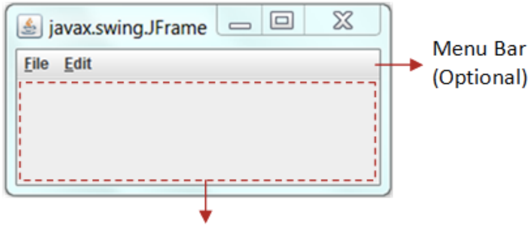
Swing Containers and Components



Object
Component
Container
1
LayoutManager
JComponent
JPanel
JScrollPane
JTabbedPane
JRootPane
JInternalFrame
JSplitPane
JLayeredPane
JOptionPane
JLabel
JComboBox
JMenuBar
JToolBar
JProgressBar
JSlider
JScrollBar
JPopupMenu
JFileChooser
JColorChooser
JTree
JTable
JList
JTextComponent
JTextField
JPasswordField
JFormattedTextField
JTextArea
JTextEditorPane
JTextPane
AbstractButton
JToggleButton
JCheckBox
JRadioButton
JButton
JMenuItem
JMenu

Containers and ContentPane

javax.swing.JFrame

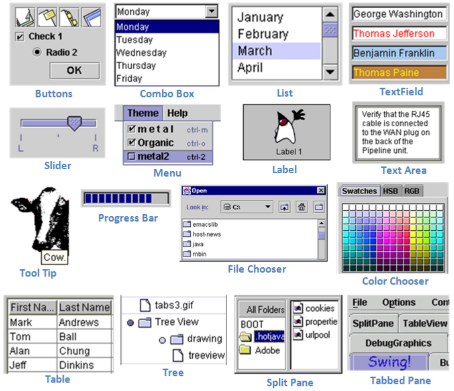


```
Container cp = aJFrame.getContentPane();
aJFrame.setContentPane(aPanel);
```

Swing components

Swing is huge (consists of 18 packages of 737 classes as in JDK 1.8) and has great depth

Compare to AWT:
12 packages of 370 classes

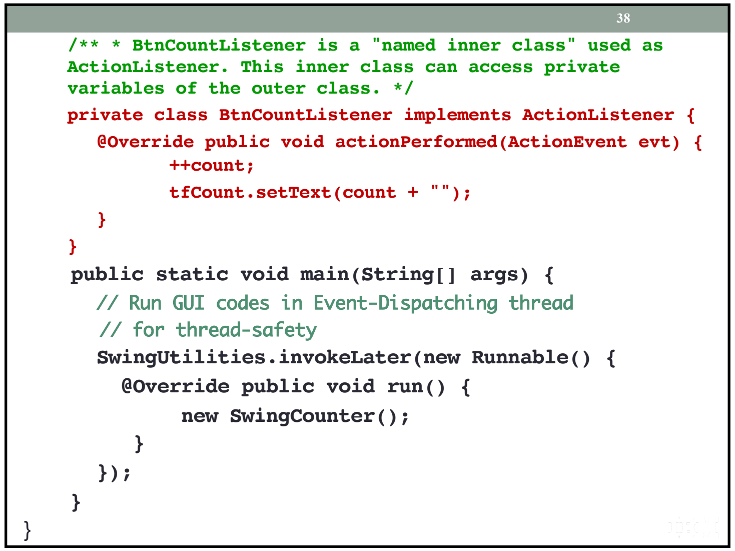


```
//A Swing GUI application inherits from top-level container
  public class SwingTemplate extends JFrame {
    // Constructor to setup the GUI components and event handlers
    public SwingTemplate() {
       // top-level content-pane from JFrame
       Container cp = getContentPane();
       cp.setLayout(new ....Layout());

       // Allocate the GUI components
       cp.add(....); //...

       // Source object adds listener
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Exit the program when the close-window button clicked
      setTitle("......");  //"super" JFrame sets title
      setSize(300, 150);   //"super" JFrame sets initial size
      setVisible(true);    // "super" JFrame shows
  }
  public static void main(String[] args) {
      new SwingTemplate();
  }
}
```

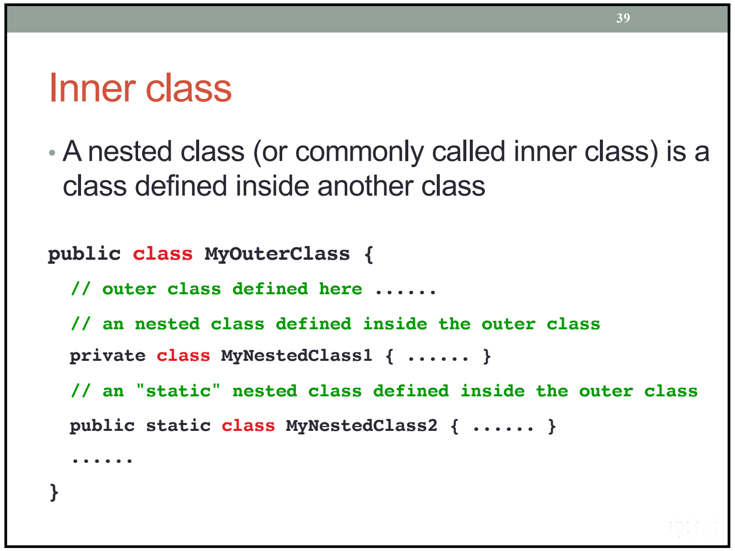
```
public class SwingCounter extends JFrame {
```

Example: SwingCounter

```
  private JTextField tfCount;
  private JButton btnCount;
  private int count = 0;
  public SwingCounter() {
    // Retrieve the content-pane of the top-level container JFrame
    // All operations done on the content-pane
    Container cp = getContentPane();   cp.setLayout(new
    FlowLayout());
    cp.add(new JLabel("Counter"));
    tfCount = new JTextField("0");
    tfCount.setEditable(false);           cp.add(tfCount);
    btnCount = new JButton("Count");      cp.add(btnCount);

    /* btnCount adds an anonymous instance of BtnCountListener (a
    named inner class) as a ActionListener */
    btnCount.addActionListener(new BtnCountListener());

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle("Swing Counter");  setSize(300, 100);
    setVisible(true);
  }
```



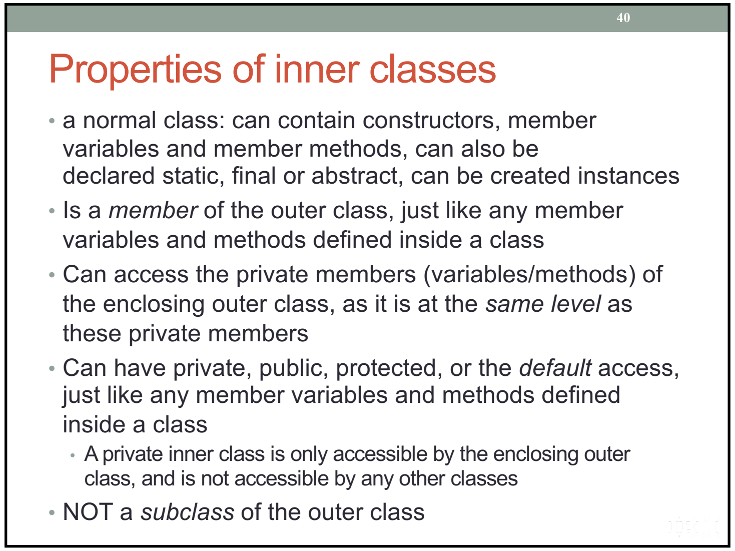
```
    /** * BtnCountListener is a "named inner class" used as
    ActionListener. This inner class can access private
    variables of the outer class. */
    private class BtnCountListener implements ActionListener {
      @Override public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + "");
      }
    }
    public static void main(String[] args) {
      // Run GUI codes in Event-Dispatching thread
      // for thread-safety
      SwingUtilities.invokeLater(new Runnable() {
        @Override public void run() {
            new SwingCounter();
        }
      });
    }
}
```



Inner class

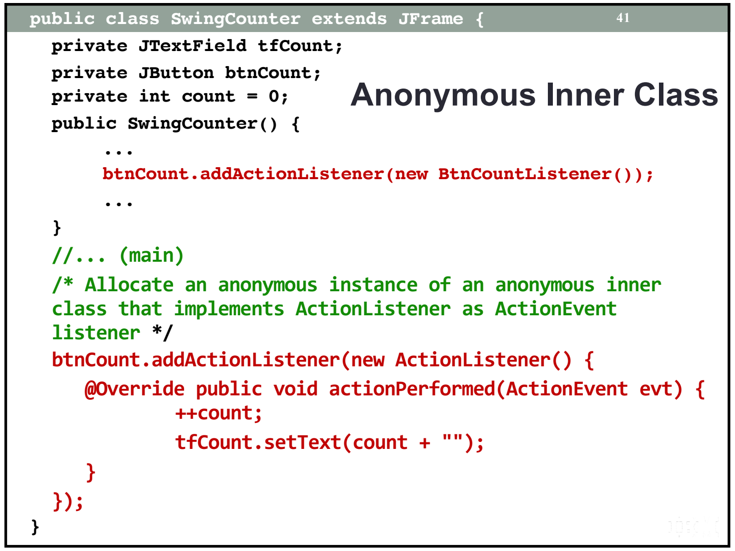
- A nested class (or commonly called inner class) is a class defined inside another class

```
public class MyOuterClass {
  // outer class defined here ......
  // an nested class defined inside the outer class
  private class MyNestedClass1 { ...... }
  // an "static" nested class defined inside the outer class
  public static class MyNestedClass2 { ...... }
  ......
}
```



Properties of inner classes

- a normal class: can contain constructors, member variables and member methods, can also be declared static, final or abstract, can be created instances
- Is a *member* of the outer class, just like any member variables and methods defined inside a class
- Can access the private members (variables/methods) of the enclosing outer class, as it is at the *same level* as these private members
- Can have private, public, protected, or the *default* access, just like any member variables and methods defined inside a class
  - A private inner class is only accessible by the enclosing outer class, and is not accessible by any other classes
- NOT a *subclass* of the outer class



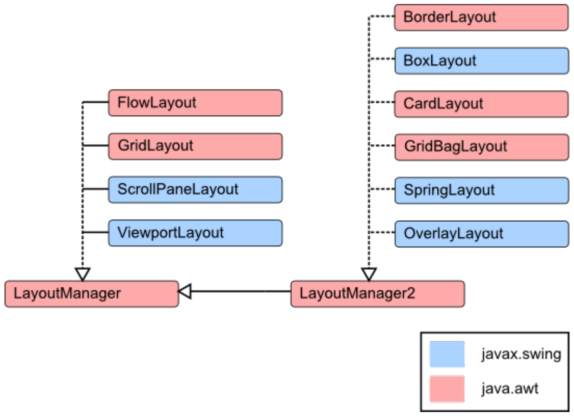
Anonymous Inner Class

```
public class SwingCounter extends JFrame {
   private JTextField tfCount;
   private JButton btnCount;
   private int count = 0;
   public SwingCounter() {
        ...
        btnCount.addActionListener(new BtnCountListener());
        ...
   }
   //... (main)
   /* Allocate an anonymous instance of an anonymous inner
   class that implements ActionListener as ActionEvent
   listener */
   btnCount.addActionListener(new ActionListener() {
      @Override public void actionPerformed(ActionEvent evt) {
              ++count;
              tfCount.setText(count + "");
      }
   });
}
```

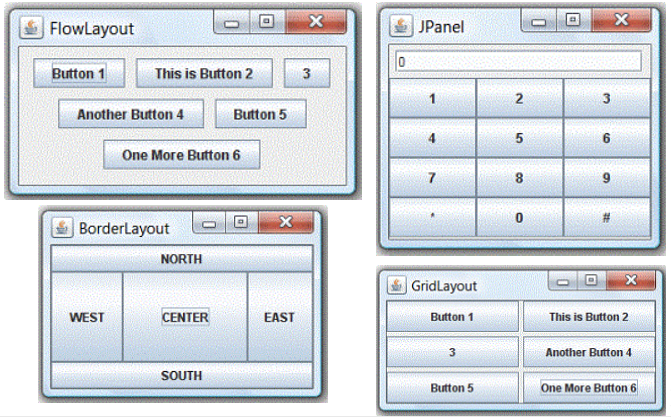
Outline

1. GUI Programming in Java
2. AWT UI Elements
3. AWT Event Handling
4. Swing GUI Elements
5. Layout Manager

AWT and Swing Layout

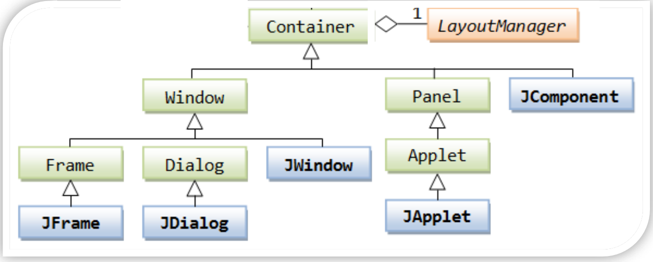


Layout management



Set layout for a container

```
// java.awt.Container
public void setLayout(LayoutManager mgr)
```



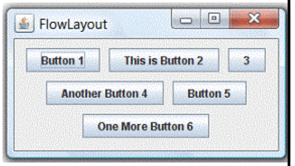
Setup a layout for a container

- Construct an instance of the chosen layout object, via new and constructor, e.g., new FlowLayout())
- Invoke the setLayout() method of the Container, with the layout object created as the argument;
- Place the GUI components into the Container using the add() method in the correct order; or into the correct zones.

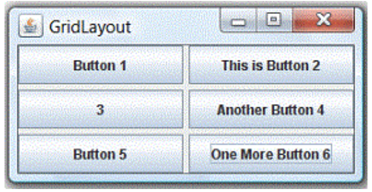
```
Panel pnl = new Panel();
// Add a new Layout object to the Panel container
pnl.setLayout(new FlowLayout());
// The Panel container adds components in a proper order
pnl.add(new JLabel("One"));
pnl.add(new JLabel("Two"));
pnl.add(new JLabel("Three"));
```

E.g. FlowLayout

```
public class AWTFlowLayoutDemo extends Frame {
  private Button btn1, btn2, btn3, btn4, btn5, btn6;
  public AWTFlowLayoutDemo () {
       // from left-to-right, and flow from top-to-bottom
       setLayout(new FlowLayout());
       btn1 = new Button("Button 1");              add(btn1);
       btn2 = new Button("This is Button 2");      add(btn2);
       btn3 = new Button("3");                     add(btn3);
       btn4 = new Button("Another Button 4");      add(btn4);
       btn5 = new Button("Button 5");              add(btn5);
       btn6 = new Button("One More Button 6");     add(btn6);

       setTitle("FlowLayout Demo");
       setSize(280, 150); setVisible(true);
  }
  public static void main(String[] args) {
       new AWTFlowLayoutDemo();
  }
}
```



E.g. GridLayout



```
...
/* set layout to 3x2
GridLayout, horizontal and vertical gaps of 3 pixels,
components are added from left-to-right, top-to-bottom */
setLayout(new GridLayout(3, 2, 3, 3));
btn1 = new Button("Button 1");              add(btn1);
btn2 = new Button("This is Button 2");      add(btn2);
btn3 = new Button("3");                     add(btn3);
btn4 = new Button("Another Button 4");      add(btn4);
btn5 = new Button("Button 5");              add(btn5);
btn6 = new Button("One More Button 6");     add(btn6);
...
```