JAVA Programming MCA 109

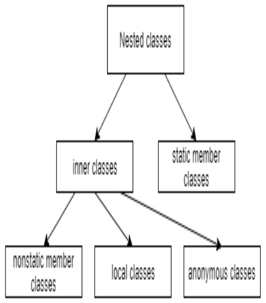
UNIT 3

Learning Objectives

- **Anonymous Classes and Inner classes in Java:** Core concept and its implementation and types of anonymous classes, nested and inner classes, and their implementation
- **Event Handling:** Different Mechanism, the Delegation Event Model, Event Classes, Event Listener Interfaces, Adapter and Inner Classes, working with windows, Graphics and Text, using AWT controls, Layout managers and menus, handling Images, animation, sound and video.
- **Swing:** Introduction to JFC (Java Foundation Classes), features of Swing, comparison with AWT, Advanced Control

Anonymous & Inner Classes

A nested class is a class that is declared inside another class or **interface.**
Anonymous classes are inner classes with no name. Since they have no name, we can't use them in order to create instances of anonymous classes. As a result, we have to declare and instantiate anonymous classes in a single expression at the point of use.

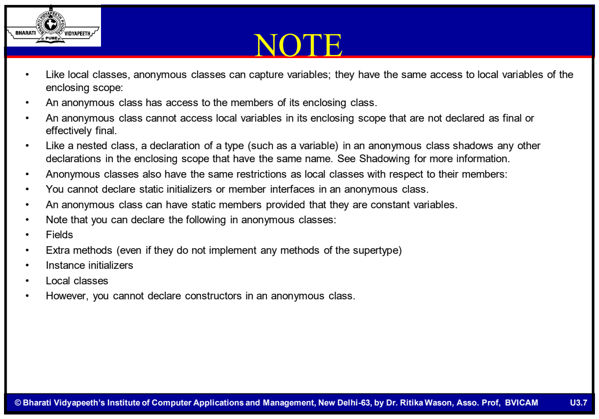


Anonymous Classes

- Anonymous classes enable you to make your code more concise.
- They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name.
- Use them if you need to use a local class only once.
- While local classes are class declarations, anonymous classes are expressions, which means that you define the class in another expression.

Example

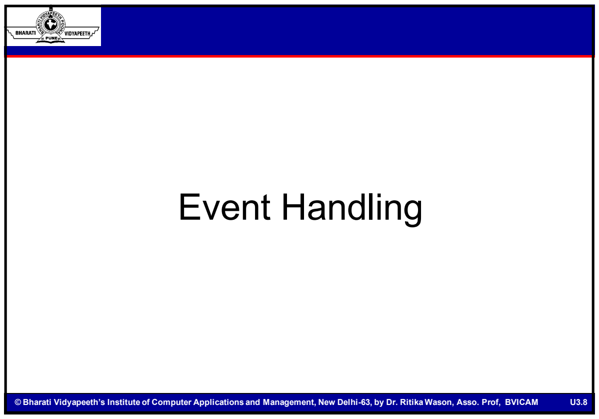
```
public class HelloWorldAnonymousClasses {public void greet();
      public void greetSomeone(String someone)   }
   public void sayHello() {
      class EnglishGreeting implements HelloWorld { String name = "world";
         public void greet() {greetSomeone("world");  }
         public void greetSomeone(String someone) {
            name = someone;
            System.out.println("Hello " + name); }}
     HelloWorld englishGreeting = new EnglishGreeting();

};
englishGreeting.greet();
public static void main(String... args) {
      HelloWorldAnonymousClasses myApp =
         new HelloWorldAnonymousClasses();
      myApp.sayHello();
   }
}
```



NOTE

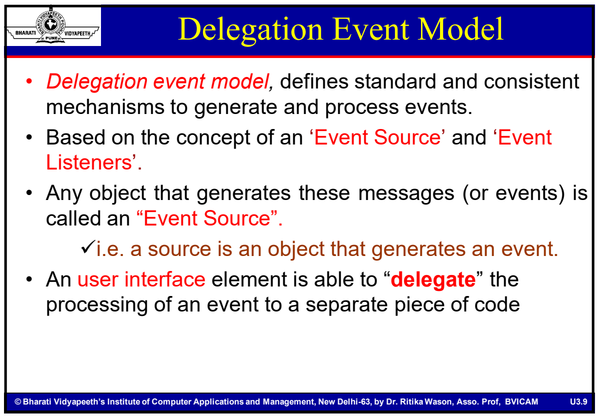
- Like local classes, anonymous classes can capture variables; they have the same access to local variables of the enclosing scope:
- An anonymous class has access to the members of its enclosing class.
- An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final.
- Like a nested class, a declaration of a type (such as a variable) in an anonymous class shadows any other declarations in the enclosing scope that have the same name. See Shadowing for more information.
- Anonymous classes also have the same restrictions as local classes with respect to their members:
- You cannot declare static initializers or member interfaces in an anonymous class.
- An anonymous class can have static members provided that they are constant variables.
- Note that you can declare the following in anonymous classes:
- Fields
- Extra methods (even if they do not implement any methods of the supertype)
- Instance initializers
- Local classes
- However, you cannot declare constructors in an anonymous class.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.7



Event Handling

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.8



Delegation Event Model

- *Delegation event model,* defines standard and consistent mechanisms to generate and process events.
- Based on the concept of an 'Event Source' and 'Event Listeners'.
- Any object that generates these messages (or events) is called an "Event Source".
  - ✓i.e. a source is an object that generates an event.
- An user interface element is able to "**delegate**" the processing of an event to a separate piece of code

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.9

Delegation Event Model

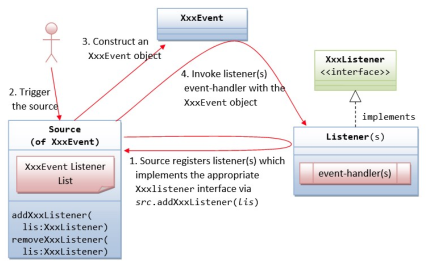
- A source must **register listeners** in order for the listeners to **receive notifications** about a specific type of event.
- Notifications are sent only to listeners that want to receive them.
- Each type of event has its own **registration method**.
- The methods that add or remove listeners are provided by the source that generates events.

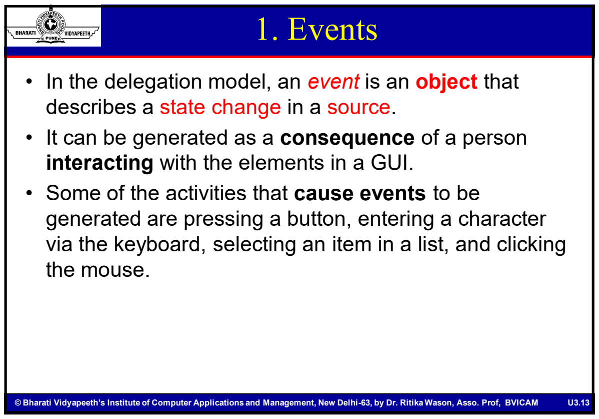
Delegation Event Model

Core Concepts:

- ✓ Events
- ✓ Event Sources
- ✓ The Event Classes
- ✓ The Event Listeners
- ✓ Explicit Event Enabling
- ✓ Adapters

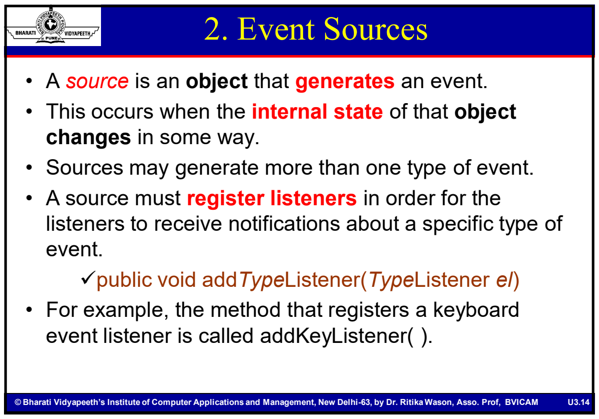
Delegation Event Model


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)



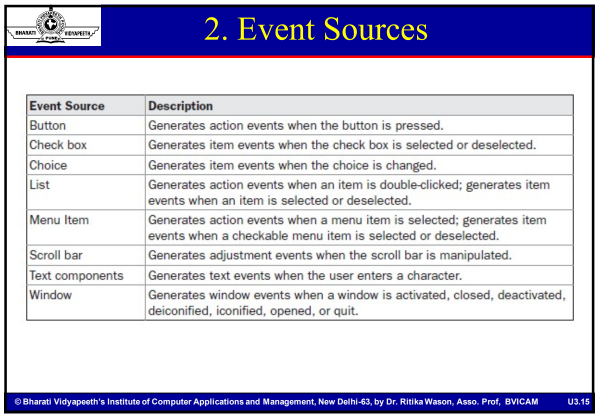
1. Events

- In the delegation model, an *event* is an **object** that describes a state change in a source.
- It can be generated as a **consequence** of a person **interacting** with the elements in a GUI.
- Some of the activities that **cause events** to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.



2. Event Sources

- A *source* is an **object** that **generates** an event.
- This occurs when the **internal state** of that **object changes** in some way.
- Sources may generate more than one type of event.
- A source must **register listeners** in order for the listeners to receive notifications about a specific type of event.
  - ✓public void add*Type*Listener(*Type*Listener *el*)
- For example, the method that registers a keyboard event listener is called addKeyListener( ).



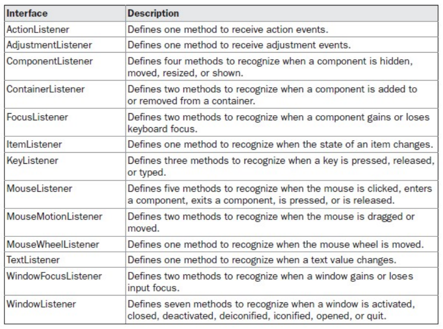
2. Event Sources

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

3. Event Listener

- A *listener* is an **object** that is **notified** when an event occurs.
- Two major requirements for a listener
  - ✓Must have been registered with one or more sources to receive notifications
  - ✓Must implement methods to receive and **process** these notifications
- For each event type that can occur, the application can add event listeners, that have methods invoked when the event occurs.
- The listeners are defined as **interfaces** in **java.awt.event**, so that an **actual listener** has to **implement** these methods.

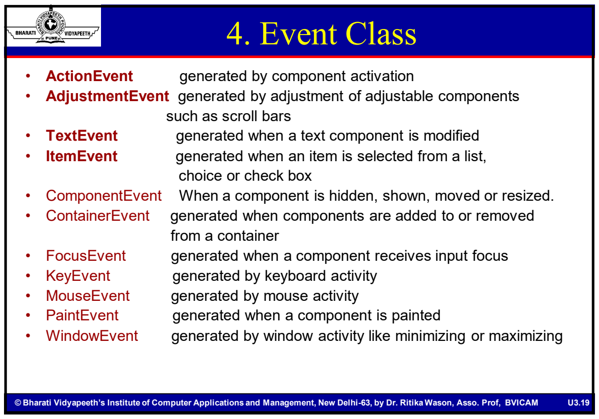
3. Event Listener Interfaces



Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

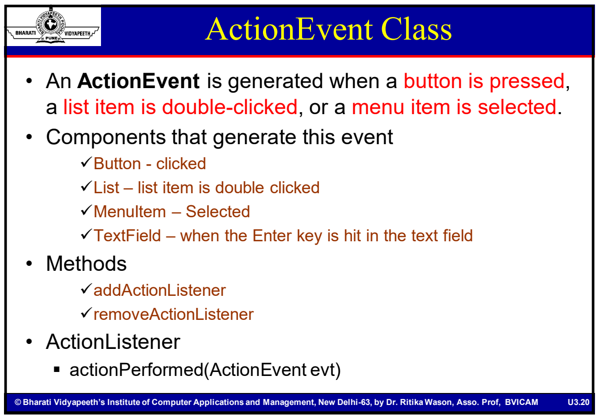
4. Event Class

- The classes that **represent events** are at the core of Java's event handling mechanism.
- At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**.
- It is the superclass for all events.



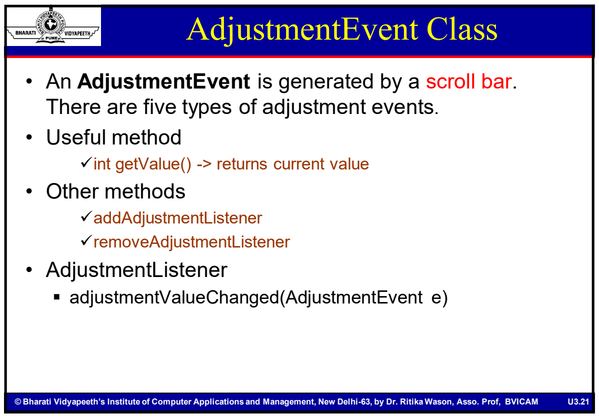
4. Event Class

• **ActionEvent**	generated by component activation
• **AdjustmentEvent**	generated by adjustment of adjustable components such as scroll bars
• **TextEvent**	generated when a text component is modified
• **ItemEvent**	generated when an item is selected from a list, choice or check box
• ComponentEvent	When a component is hidden, shown, moved or resized.
• ContainerEvent	generated when components are added to or removed from a container
• FocusEvent	generated when a component receives input focus
• KeyEvent	generated by keyboard activity
• MouseEvent	generated by mouse activity
• PaintEvent	generated when a component is painted
• WindowEvent	generated by window activity like minimizing or maximizing



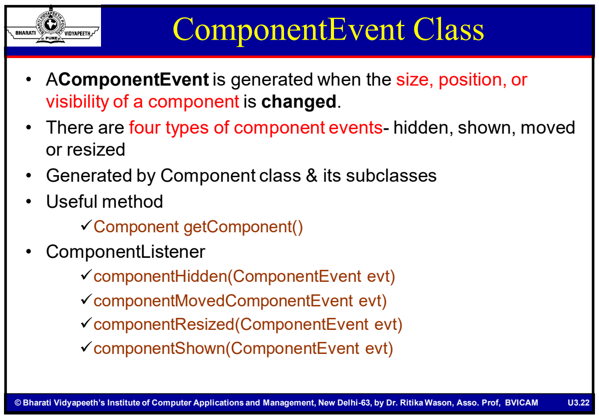
ActionEvent Class

- An **ActionEvent** is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
- Components that generate this event
  - ✓Button - clicked
  - ✓List – list item is double clicked
  - ✓MenuItem – Selected
  - ✓TextField – when the Enter key is hit in the text field
- Methods
  - ✓addActionListener
  - ✓removeActionListener
- ActionListener
  - actionPerformed(ActionEvent evt)



AdjustmentEvent Class

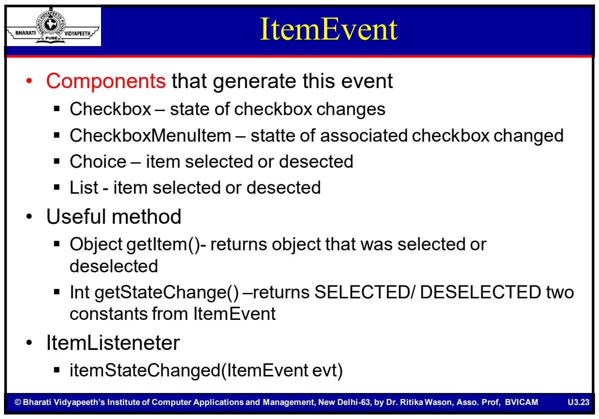
- An **AdjustmentEvent** is generated by a scroll bar. There are five types of adjustment events.
- Useful method
  - ✓int getValue() -> returns current value
- Other methods
  - ✓addAdjustmentListener
  - ✓removeAdjustmentListener
- AdjustmentListener
  - adjustmentValueChanged(AdjustmentEvent e)



ComponentEvent Class

- A **ComponentEvent** is generated when the size, position, or visibility of a component is **changed**.
- There are four types of component events- hidden, shown, moved or resized
- Generated by Component class & its subclasses
- Useful method
  - ✓ Component getComponent()
- ComponentListener
  - ✓ componentHidden(ComponentEvent evt)
  - ✓ componentMovedComponentEvent evt)
  - ✓ componentResized(ComponentEvent evt)
  - ✓ componentShown(ComponentEvent evt)

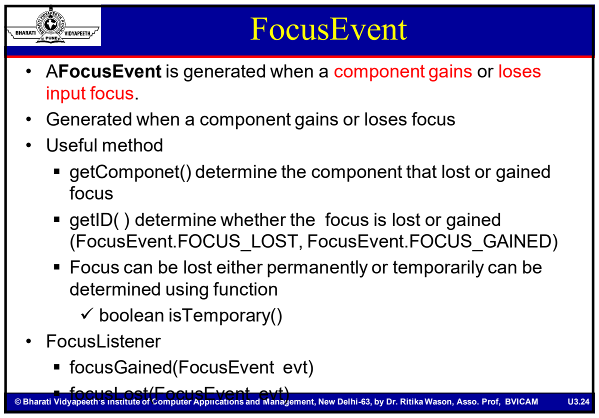
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.22



ItemEvent

- Components that generate this event
  - Checkbox – state of checkbox changes
  - CheckboxMenuItem – statte of associated checkbox changed
  - Choice – item selected or desected
  - List - item selected or desected
- Useful method
  - Object getItem()- returns object that was selected or deselected
  - Int getStateChange() –returns SELECTED/ DESELECTED two constants from ItemEvent
- ItemListeneter
  - itemStateChanged(ItemEvent evt)

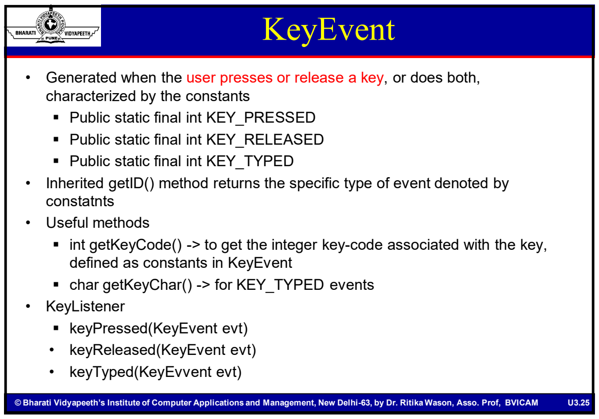
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.23



FocusEvent

- A **FocusEvent** is generated when a component gains or loses input focus.
- Generated when a component gains or loses focus
- Useful method
  - getComponet() determine the component that lost or gained focus
  - getID( ) determine whether the focus is lost or gained (FocusEvent.FOCUS_LOST, FocusEvent.FOCUS_GAINED)
  - Focus can be lost either permanently or temporarily can be determined using function
    - ✓ boolean isTemporary()
- FocusListener
  - focusGained(FocusEvent evt)
  - focusLost(FocusEvent evt)

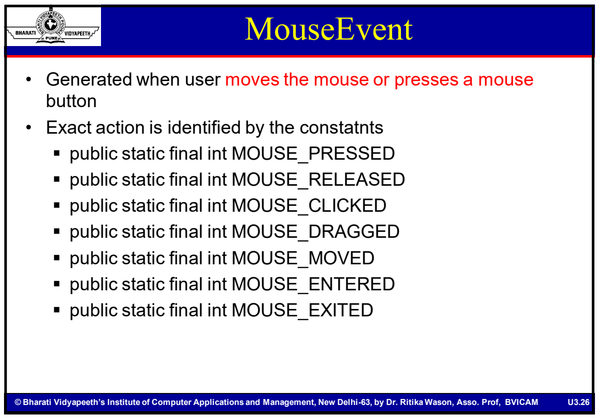
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.24



KeyEvent

- Generated when the user presses or release a key, or does both, characterized by the constants
  - Public static final int KEY_PRESSED
  - Public static final int KEY_RELEASED
  - Public static final int KEY_TYPED
- Inherited getID() method returns the specific type of event denoted by constatnts
- Useful methods
  - int getKeyCode() -> to get the integer key-code associated with the key, defined as constants in KeyEvent
  - char getKeyChar() -> for KEY_TYPED events
- KeyListener
  - keyPressed(KeyEvent evt)
  - keyReleased(KeyEvent evt)
  - keyTyped(KeyEvvent evt)

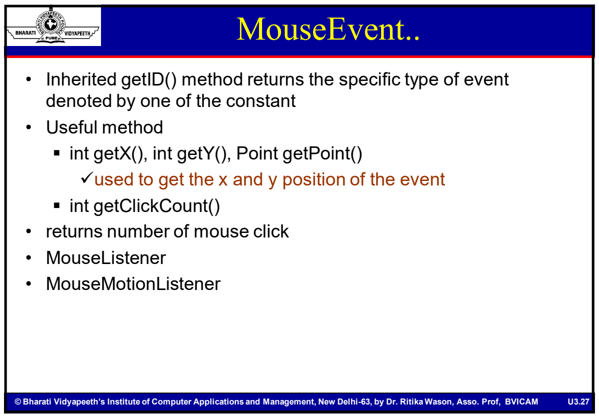
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.25



MouseEvent

- Generated when user moves the mouse or presses a mouse button
- Exact action is identified by the constatnts
  - public static final int MOUSE_PRESSED
  - public static final int MOUSE_RELEASED
  - public static final int MOUSE_CLICKED
  - public static final int MOUSE_DRAGGED
  - public static final int MOUSE_MOVED
  - public static final int MOUSE_ENTERED
  - public static final int MOUSE_EXITED

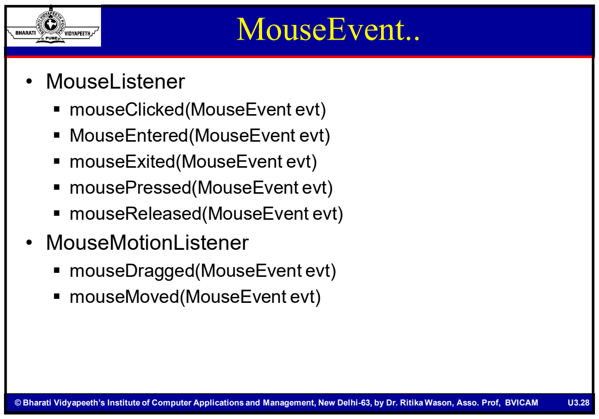
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.26



MouseEvent..

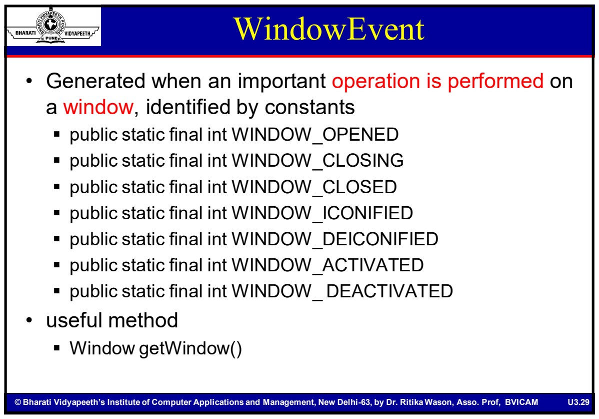
- Inherited getID() method returns the specific type of event denoted by one of the constant
- Useful method
  - int getX(), int getY(), Point getPoint()
    - ✓used to get the x and y position of the event
  - int getClickCount()
- returns number of mouse click
- MouseListener
- MouseMotionListener

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.27



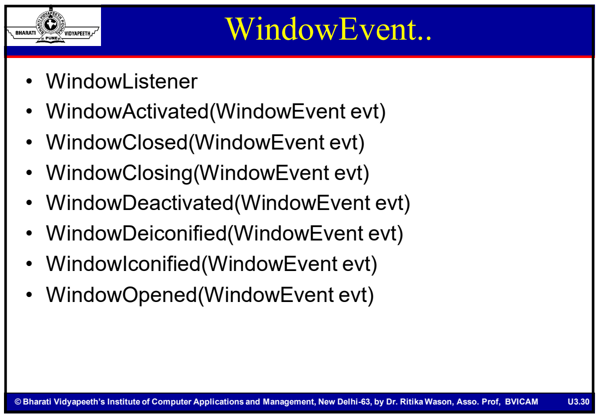
MouseEvent..

- MouseListener
  - mouseClicked(MouseEvent evt)
  - MouseEntered(MouseEvent evt)
  - mouseExited(MouseEvent evt)
  - mousePressed(MouseEvent evt)
  - mouseReleased(MouseEvent evt)
- MouseMotionListener
  - mouseDragged(MouseEvent evt)
  - mouseMoved(MouseEvent evt)



WindowEvent

- Generated when an important operation is performed on a window, identified by constants
  - public static final int WINDOW_OPENED
  - public static final int WINDOW_CLOSING
  - public static final int WINDOW_CLOSED
  - public static final int WINDOW_ICONIFIED
  - public static final int WINDOW_DEICONIFIED
  - public static final int WINDOW_ACTIVATED
  - public static final int WINDOW_ DEACTIVATED
- useful method
  - Window getWindow()



WindowEvent..

- WindowListener
- WindowActivated(WindowEvent evt)
- WindowClosed(WindowEvent evt)
- WindowClosing(WindowEvent evt)
- WindowDeactivated(WindowEvent evt)
- WindowDeiconified(WindowEvent evt)
- WindowIconified(WindowEvent evt)
- WindowOpened(WindowEvent evt)

5. EventAdapter Class

- An adapter class provides an **empty implementation** of **all methods** in an event listener interface.
- Java.awt.event package defines an adapter class to each low level listener interface
- Adapter classes are useful when you want to **receive** and **process** only some of the events that are handled by a particular event listener interface.
- An event adapter **implements stubs** for all the methods of the corresponding interface
- A listener can subclass the adapter and override only stub method of interest

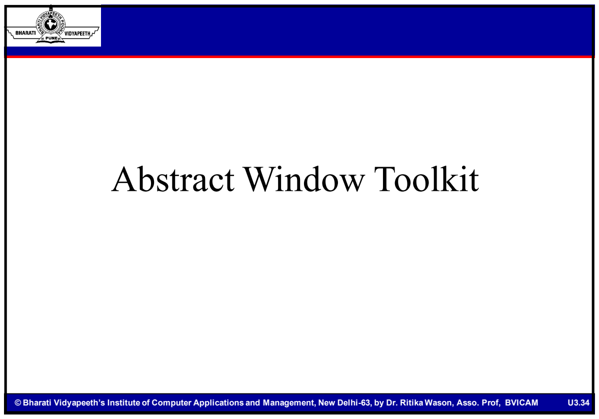
5. EventAdapter..

ComponentListener	ComponentAdapter
ContainerLitener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

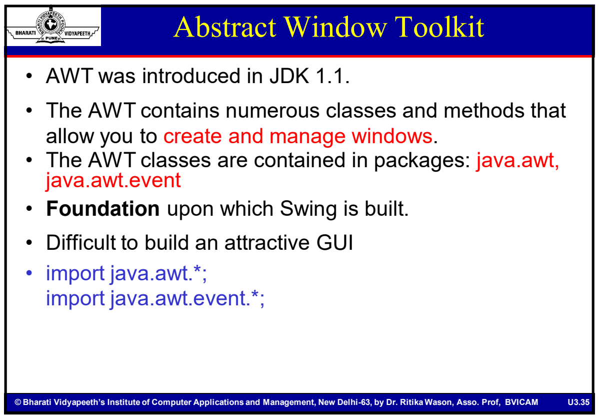
Using Event Delegation Model

1. Implement the appropriate interface in the **listener** so that it will receive the **type of event desired**.
2. Implement code to register and unregister (if necessary) the **listener** as a recipient for the event notifications.

Remember that a source may generate several types of events. Each event must be registered separately.

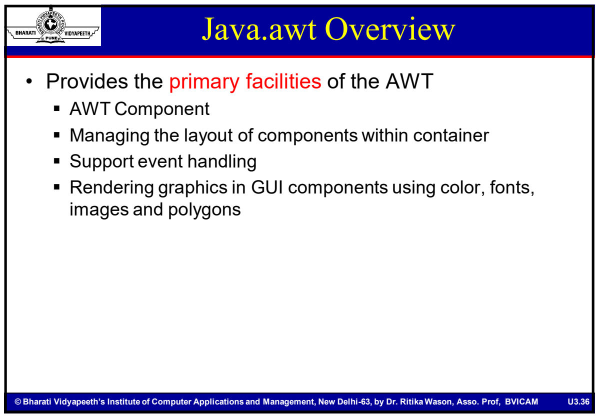


Abstract Window Toolkit



Abstract Window Toolkit

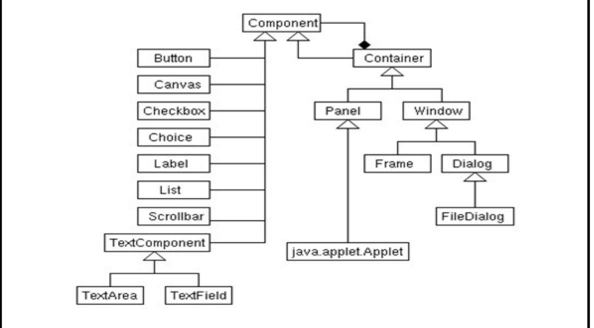
- AWT was introduced in JDK 1.1.
- The AWT contains numerous classes and methods that allow you to create and manage windows.
- The AWT classes are contained in packages: java.awt, java.awt.event
- **Foundation** upon which Swing is built.
- Difficult to build an attractive GUI
- import java.awt.*;
  import java.awt.event.*;



Java.awt Overview

- Provides the primary facilities of the AWT
  - AWT Component
  - Managing the layout of components within container
  - Support event handling
  - Rendering graphics in GUI components using color, fonts, images and polygons

AWT Class Hierarchy



Containers and Components

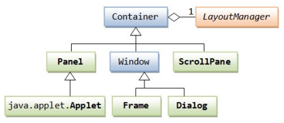
- The job of a Container is to hold and display Components
- Some common subclasses of Component are Button, Checkbox, Label, Scrollbar, TextField, **and** TextArea
- A Container is also a Component
  - This allows Containers to be nested
- Some Container subclasses are Panel (and Applet), Window, and Frame

Component

- Component is the **superclass** of most of the displayable classes defined within the AWT. Note: it is **abstract**.
- Menu Component is another class which is similar to Component except it is the superclass for all GUI items which can be displayed within a drop-down menu.
- The Component class defines data and methods which are relevant to all Components
  - ✓setBounds
  - ✓setSize
  - ✓setLocation
  - ✓setFont
  - ✓setEnabled
  - ✓setVisible
  - ✓setForeground -- colour
  - ✓setBackground -- colour

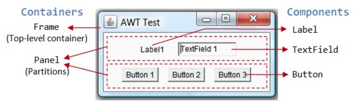
Container

- Subclass of Component.
- Contains components
- For a component to be placed on the screen, it must be placed
- within a Container
- The Container class defined all the data and methods necessary for managing groups of Components
  - ✓add
  - ✓getComponent
  - ✓getMaximumSize
  - ✓getMinimumSize
  - ✓getPreferredSize
  - ✓remove
  - ✓removeAll



Panel

- Concrete subclass of the Container class
- Doesn't have a title, menus or borders
- Ideal for packing other components and panels to build component hierarchies using inherited add() method



Window

•Represents top level window that has no title, menus or borders

•Top level window can't be incorporated into other components

•void pack() method initiates the layout manager

•void show() used to make the window visible and bring it to the front of any other windows

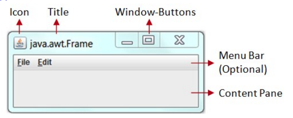
•Windows are initially hidden unlike other components

•void dispose() used to free the windowing resources

Frame

- User resizable and movable top-level window that can have a title-bar, an icon and menus
- Can be root of a component hierarchy
- Two constructors
- Frame()/ Frame(String title)

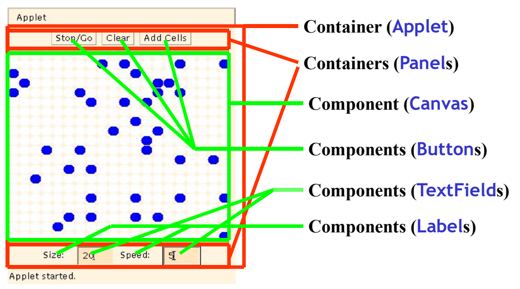
```
{
    Frame guiFrame = new Frame("my frame");
    guiFrame.add(new Button("OK"));
    guiFrame.setSize(200,300);
    guiFrame.pack();
    guiFrame.setVisible(true);
}
```



Dialog

- Defines an independent, user resizable window that can have a title-bar and a border
- Serves as a container
- Can be root of a component hierarchy
- Can be modal/ non-modal
- Dialog constructors
- Dialog(Frame parent) / Dialog(Frame parent, Boolean modal) / Dialog(Frame parent, String title) / Dialog(Frame parent, String title, Boolean modal)

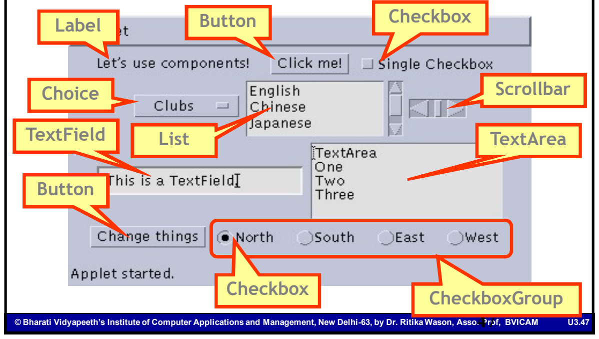
Example



GUI control components

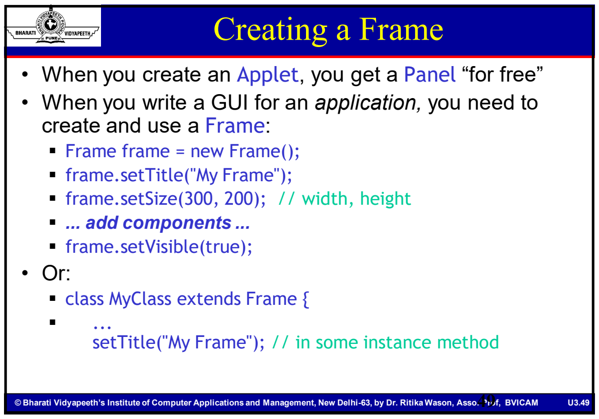
- Primary elements of GUI
- Concrete subclasses of the Component class
- Essential steps in making use of a GUI control component:
  - Create a component using constructor
    - *Button b = new Button("OK");*
  - Add to a container using a layout manager
    - *guiFrame.add(guiComponent)*
  - Register Listner with the GUI component to receive events.

Some types of components



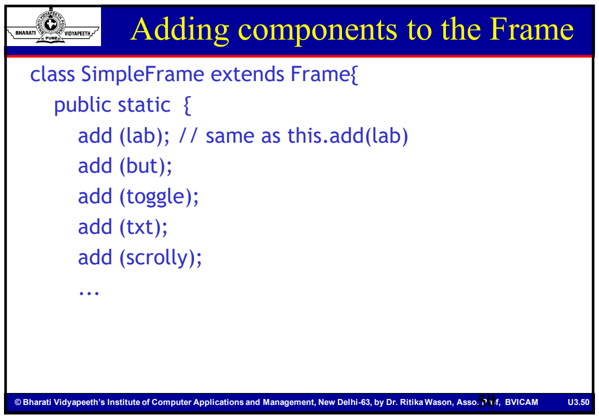
Creating Components

```
Label lab = new Label ("Hi, Dave!");
Button but = new Button ("Click me!");
Checkbox toggle = new Checkbox ("toggle");
TextField txt =
   new TextField ("Initial text.", 20);
Scrollbar scrolly = new Scrollbar
  (Scrollbar.HORIZONTAL, initialValue,
   bubbleSize, minValue, maxValue);
```



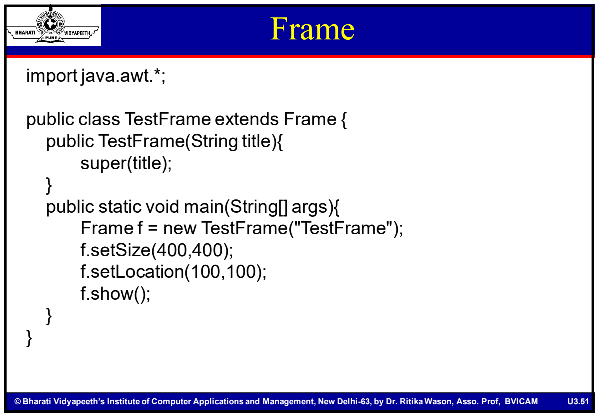
Creating a Frame

- When you create an Applet, you get a Panel "for free"
- When you write a GUI for an *application,* you need to create and use a Frame:
  - Frame frame = new Frame();
  - frame.setTitle("My Frame");
  - frame.setSize(300, 200); // width, height
  - ***... add components ...***
  - frame.setVisible(true);
- Or:
  - class MyClass extends Frame {
  - ...
    setTitle("My Frame"); // in some instance method



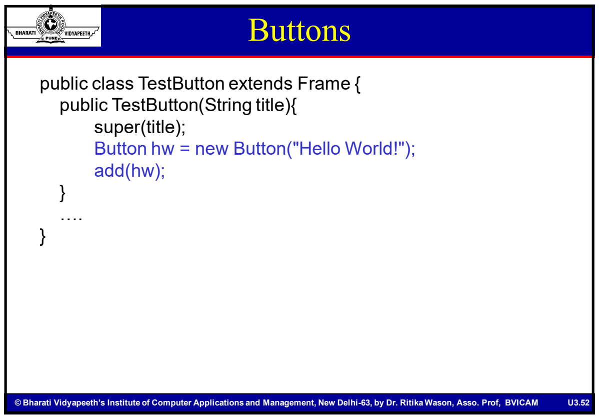
Adding components to the Frame

```
class SimpleFrame extends Frame{
   public static  {
      add (lab); // same as this.add(lab)
      add (but);
      add (toggle);
      add (txt);
      add (scrolly);
      ...
```



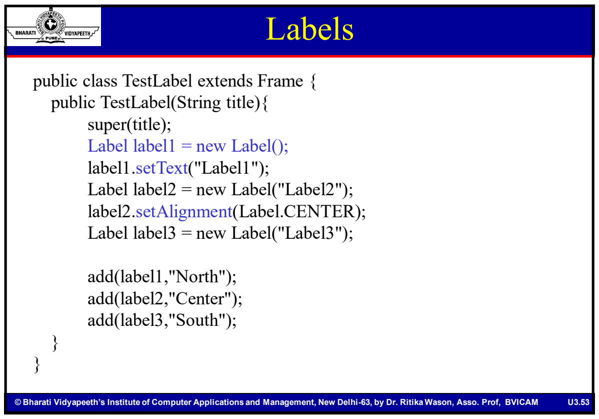
Frame

```
import java.awt.*;

public class TestFrame extends Frame {
   public TestFrame(String title){
         super(title);
   }
   public static void main(String[] args){
         Frame f = new TestFrame("TestFrame");
         f.setSize(400,400);
         f.setLocation(100,100);
         f.show();
   }
}
```



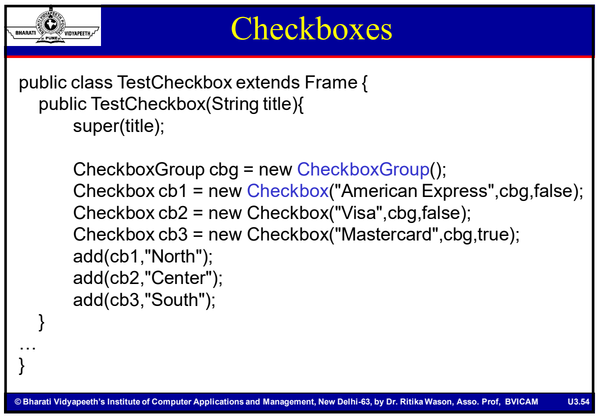
Buttons

```
public class TestButton extends Frame {
   public TestButton(String title){
        super(title);
        Button hw = new Button("Hello World!");
        add(hw);
   }
   ….
}
```



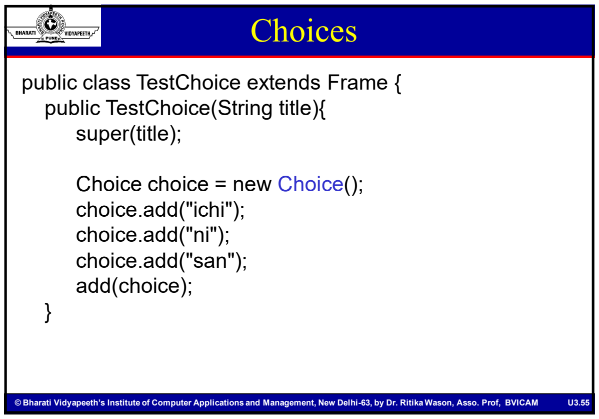
Labels

```
public class TestLabel extends Frame {
   public TestLabel(String title){
        super(title);
        Label label1 = new Label();
        label1.setText("Label1");
        Label label2 = new Label("Label2");
        label2.setAlignment(Label.CENTER);
        Label label3 = new Label("Label3");

        add(label1,"North");
        add(label2,"Center");
        add(label3,"South");
   }
}
```



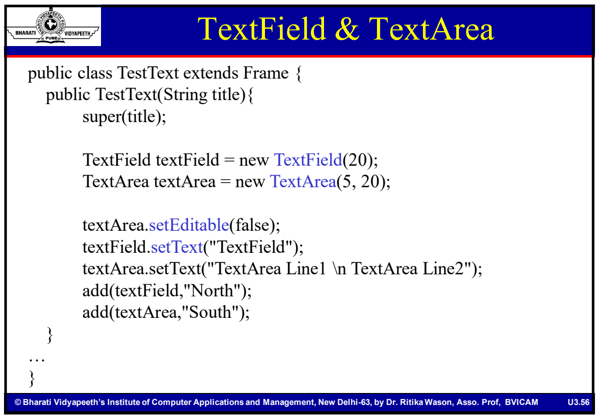
Checkboxes

```
public class TestCheckbox extends Frame {
   public TestCheckbox(String title){
        super(title);

        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox cb1 = new Checkbox("American Express",cbg,false);
        Checkbox cb2 = new Checkbox("Visa",cbg,false);
        Checkbox cb3 = new Checkbox("Mastercard",cbg,true);
        add(cb1,"North");
        add(cb2,"Center");
        add(cb3,"South");
   }
…
}
```



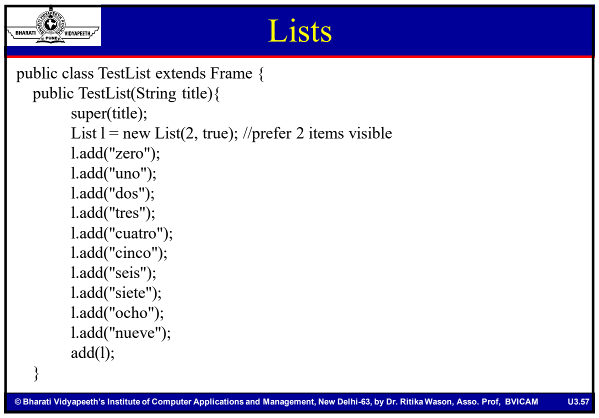
Choices

```
public class TestChoice extends Frame {
   public TestChoice(String title){
       super(title);

       Choice choice = new Choice();
       choice.add("ichi");
       choice.add("ni");
       choice.add("san");
       add(choice);
   }
```



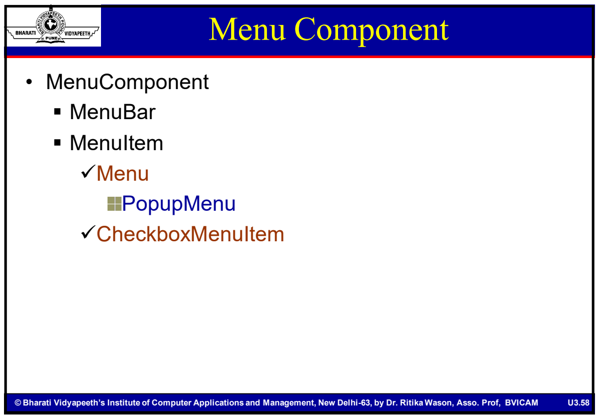
TextField & TextArea

```
public class TestText extends Frame {
   public TestText(String title){
         super(title);

         TextField textField = new TextField(20);
         TextArea textArea = new TextArea(5, 20);

         textArea.setEditable(false);
         textField.setText("TextField");
         textArea.setText("TextArea Line1 \n TextArea Line2");
         add(textField,"North");
         add(textArea,"South");
   }
…
}
```



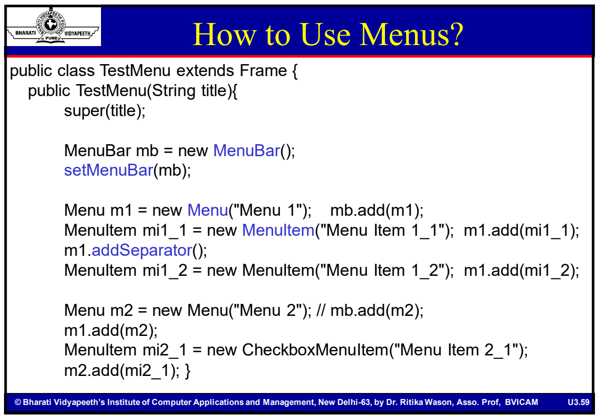
Lists

```
public class TestList extends Frame {
   public TestList(String title){
         super(title);
         List l = new List(2, true); //prefer 2 items visible
         l.add("zero");
         l.add("uno");
         l.add("dos");
         l.add("tres");
         l.add("cuatro");
         l.add("cinco");
         l.add("seis");
         l.add("siete");
         l.add("ocho");
         l.add("nueve");
         add(l);
   }
```



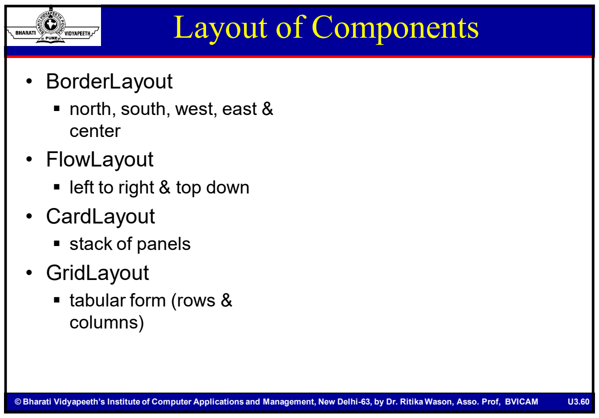
Menu Component

- MenuComponent
  - MenuBar
  - MenuItem
    - ✓Menu
      - PopupMenu
    - ✓CheckboxMenuItem



How to Use Menus?

```
public class TestMenu extends Frame {
   public TestMenu(String title){
         super(title);

         MenuBar mb = new MenuBar();
         setMenuBar(mb);

         Menu m1 = new Menu("Menu 1");    mb.add(m1);
         MenuItem mi1_1 = new MenuItem("Menu Item 1_1");  m1.add(mi1_1);
         m1.addSeparator();
         MenuItem mi1_2 = new MenuItem("Menu Item 1_2");  m1.add(mi1_2);

         Menu m2 = new Menu("Menu 2"); // mb.add(m2);
         m1.add(m2);
         MenuItem mi2_1 = new CheckboxMenuItem("Menu Item 2_1");
         m2.add(mi2_1); }
```

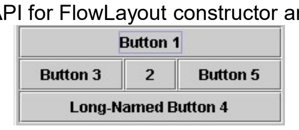


Layout of Components

- BorderLayout
  - north, south, west, east & center
- FlowLayout
  - left to right & top down
- CardLayout
  - stack of panels
- GridLayout
  - tabular form (rows & columns)

1. Flow Layout

- The components are **arranged** in the container from left to right in the order in which they **were added**. When one row becomes filled, a new row is started.
- FlowLayout is the default Layout Manager for **Applets** and **Panels**.
- Look at the API for FlowLayout constructor and the add method

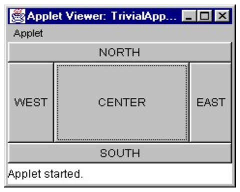


Flow Layout..

```
import java.awt.*;
import java.awt.event.*;
public class SimpleFrame1
{
public static void main (String[] args)
{
    Frame frame =new Frame("test me");
    frame.setLayout(new FlowLayout());
    Button button1 = new Button("button1");
    frame.add(button1);
    Button button2 = new Button("button2");
    frame.add(button2);
    Button button3 = new Button("button3");
    frame.add(button3);
    …
```

2. Border Layout

- At most five components can be added
- If you want more components, add a Panel, then add components to it.
- setLayout (new BorderLayout());



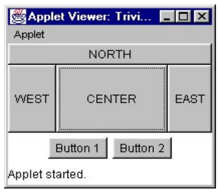
add (new Button("NORTH"), BorderLayout.NORTH);

Border Layout with five Buttons

```
{
  setLayout (new BorderLayout ());
  add (new Button ("NORTH"), BorderLayout.NORTH);
  add (new Button ("SOUTH"), BorderLayout.SOUTH);
  add (new Button ("EAST"), BorderLayout.EAST);
  add (new Button ("WEST"), BorderLayout.WEST);
  add (new Button ("CENTER"), BorderLayout.CENTER);
}
```

Using a Panel

```
Panel p = new Panel();
add (p, BorderLayout.SOUTH);
p.add (new Button ("Button 1"));
p.add (new Button ("Button 2"));
```

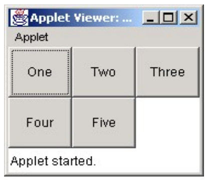


Applet started.

3. Grid Layout

- The GridLayout manager divides the container up into a given number of rows and columns:

  new GridLayout(*rows*, *columns*)

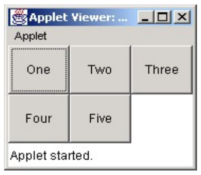


Applet started.

- All sections of the grid are equally sized and as large as possible

Complete example: GridLayout

```
setLayout(new GridLayout(2, 3));
add(new Button("One"));
add(new Button("Two"));
add(new Button("Three"));
add(new Button("Four"));
add(new Button("Five"));
```

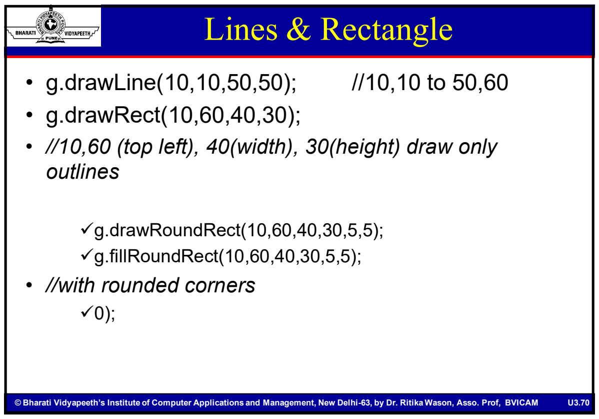


5. Card Layout Manager

- In a CardLayout only one of its components is **visible** at any given time.
- Think of the components as a set of "cards". Only one card is visible at a time, but you can **flip** from one card to another.
- Methods are provided in the CardLayout class for flipping to the first card, to the next card, and to the last card in the deck.

Graphics in JAVA

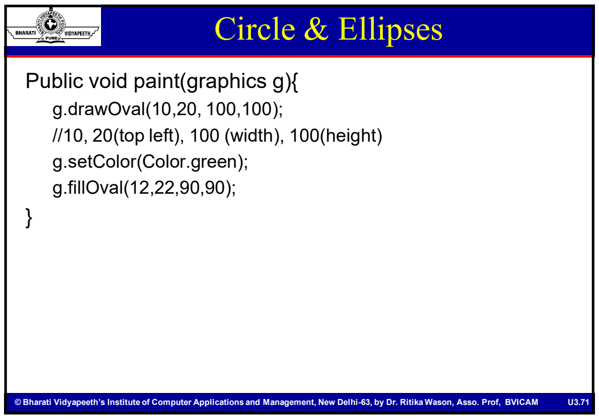
- Most important feature
- JAVA's coordinate System
  - Origin(0,0) in the upper left corner
- **Graphic Class**
  - ✓Includes methods for drawing different types of shapes
  - ✓Argument' represents end points, corner or starting locations of a shape



Lines & Rectangle

- g.drawLine(10,10,50,50); //10,10 to 50,60
- g.drawRect(10,60,40,30);
- *//10,60 (top left), 40(width), 30(height) draw only outlines*
  - ✓g.drawRoundRect(10,60,40,30,5,5);
  - ✓g.fillRoundRect(10,60,40,30,5,5);
- *//with rounded corners*
  - ✓0);

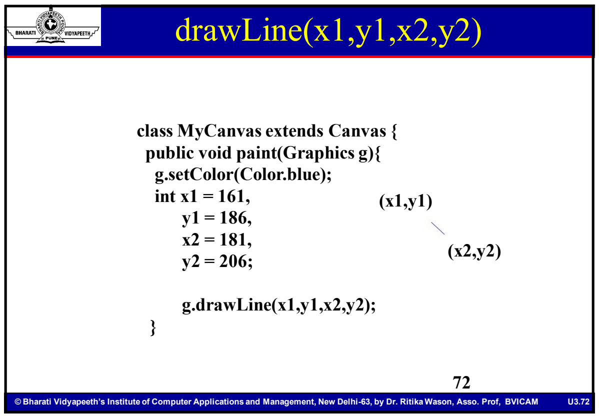
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.70



Circle & Ellipses

```
Public void paint(graphics g){
    g.drawOval(10,20, 100,100);
    //10, 20(top left), 100 (width), 100(height)
    g.setColor(Color.green);
    g.fillOval(12,22,90,90);
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.71



drawLine(x1,y1,x2,y2)

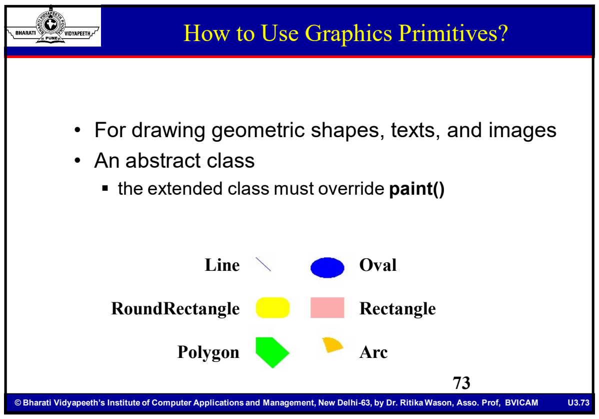
```
class MyCanvas extends Canvas {
  public void paint(Graphics g){
    g.setColor(Color.blue);
    int x1 = 161,
        y1 = 186,
        x2 = 181,
        y2 = 206;

        g.drawLine(x1,y1,x2,y2);
  }
```

(x1,y1)

(x2,y2)

72

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.72

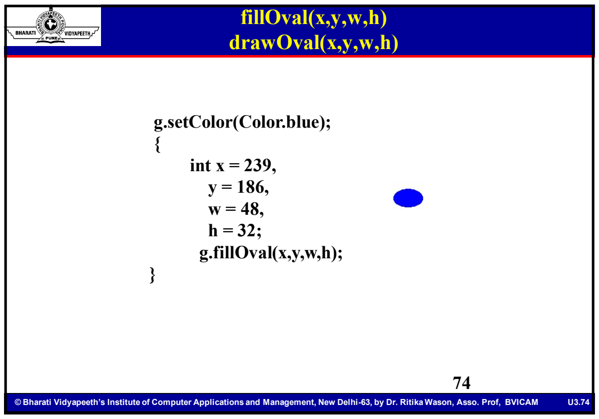


How to Use Graphics Primitives?

- For drawing geometric shapes, texts, and images
- An abstract class
  - the extended class must override **paint()**

Line
Oval
RoundRectangle
Rectangle
Polygon
Arc

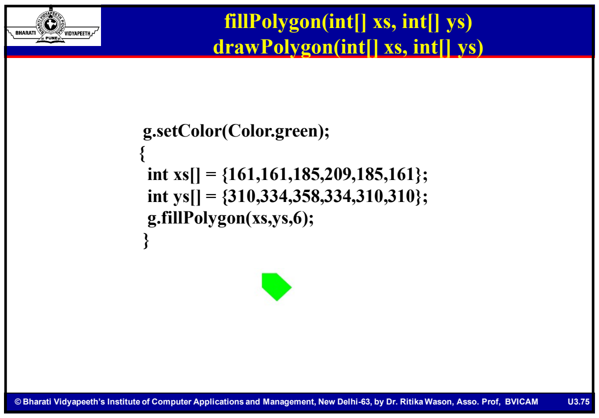
73



fillOval(x,y,w,h) drawOval(x,y,w,h)

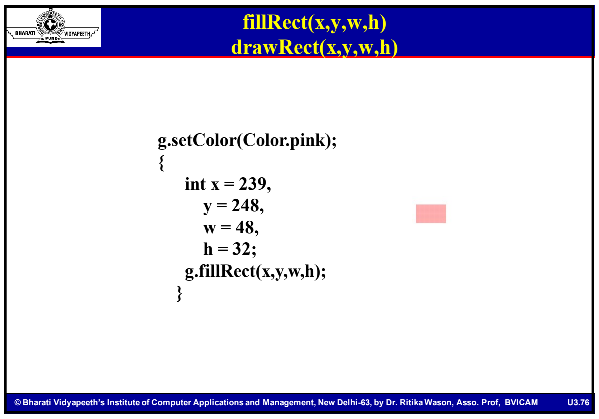
```
g.setColor(Color.blue);
{
      int x = 239,
         y = 186,
         w = 48,
         h = 32;
       g.fillOval(x,y,w,h);
}
```

74



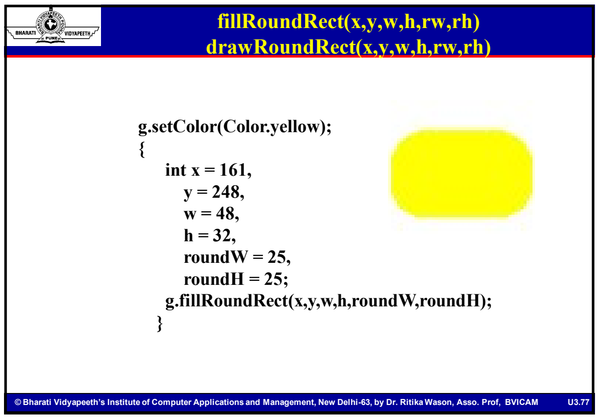
fillPolygon(int[] xs, int[] ys) drawPolygon(int[] xs, int[] ys)

```
g.setColor(Color.green);
{
 int xs[] = {161,161,185,209,185,161};
 int ys[] = {310,334,358,334,310,310};
 g.fillPolygon(xs,ys,6);
}
```



fillRect(x,y,w,h)
drawRect(x,y,w,h)

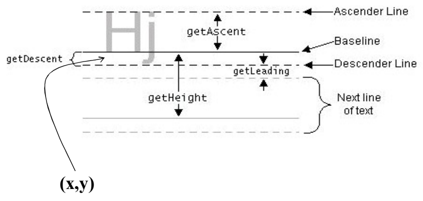
```
g.setColor(Color.pink);
{
    int x = 239,
        y = 248,
        w = 48,
        h = 32;
    g.fillRect(x,y,w,h);
  }
```

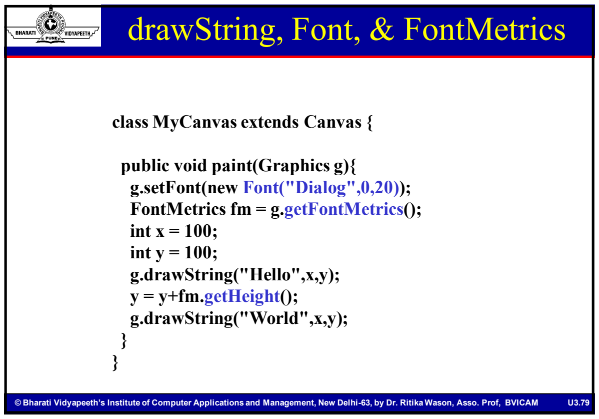


fillRoundRect(x,y,w,h,rw,rh)
drawRoundRect(x,y,w,h,rw,rh)

```
g.setColor(Color.yellow);
{
    int x = 161,
        y = 248,
        w = 48,
        h = 32,
        roundW = 25,
        roundH = 25;
    g.fillRoundRect(x,y,w,h,roundW,roundH);
  }
```

drawString(s,x,y)
FontMetrics

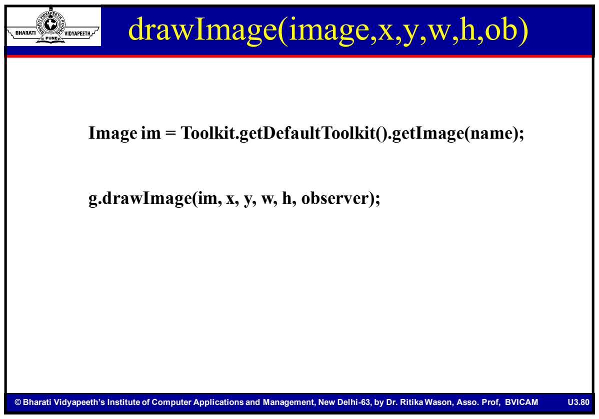




drawString, Font, & FontMetrics

```
class MyCanvas extends Canvas {

  public void paint(Graphics g){
    g.setFont(new Font("Dialog",0,20));
    FontMetrics fm = g.getFontMetrics();
    int x = 100;
    int y = 100;
    g.drawString("Hello",x,y);
    y = y+fm.getHeight();
    g.drawString("World",x,y);
  }
}
```

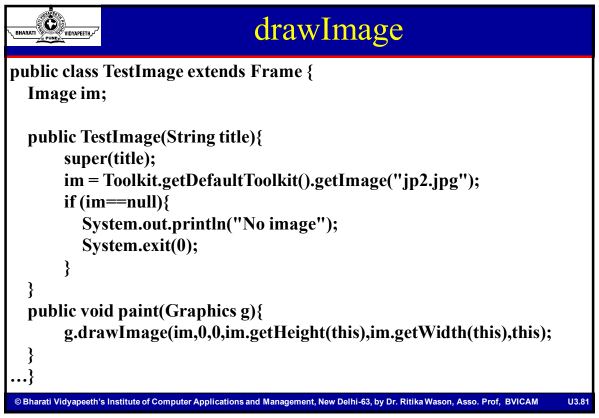
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.79



drawImage(image,x,y,w,h,ob)

```
Image im = Toolkit.getDefaultToolkit().getImage(name);


g.drawImage(im, x, y, w, h, observer);
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.80



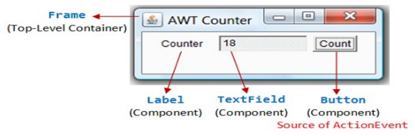
drawImage

```
public class TestImage extends Frame {
   Image im;

   public TestImage(String title){
         super(title);
         im = Toolkit.getDefaultToolkit().getImage("jp2.jpg");
         if (im==null){
             System.out.println("No image");
             System.exit(0);
         }
   }
   public void paint(Graphics g){
         g.drawImage(im,0,0,im.getHeight(this),im.getWidth(this),this);
   }
...}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.81

Exercise

- Write an AWT GUI application (called AWTCounter) Each time the "Count" button is clicked, the counter value shall increase by 1.
- The program has three components:
  - a Label "Counter";
  - a non-editable TextField to display the counter value; and
  - a Button "Count".
- Swing



About JFC and Swing

- JFC – Java^{TM} Foundation Classes
- Encompass a group of features for constructing graphical user interfaces (GUI).
- Implemented without any native code.
- "Swing" is the codename of the project that developed the first JFC components (JFC 1.1[1]).
- The name "Swing" is frequently used to refer to new components and related API.

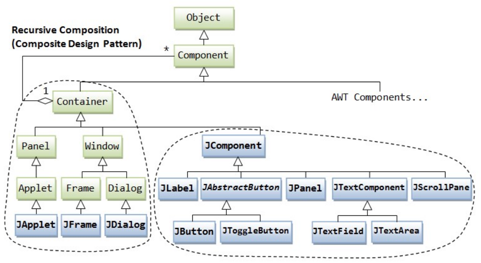
To build a GUI

- Make somewhere to display things--a Frame, a Window, or an Applet
- Create controls (buttons, text areas, etc.)
- Add your controls to your display area
- Arrange, or *layout*, your controls
- Attach Listeners actions to your controls
  - Interacting with a Component causes an Event to occur
  - A Listener gets a message when an interesting event occurs, and executes some code to deal with it

Swing

- Same concepts as AWT
- Doesn't work in ancient Java implementations (Java 1.1 and earlier)
- Many more controls, and they are more flexible
  - Some controls, but not all, are a lot more complicated
- Gives a choice of "look and feel" packages
- Much easier to build an attractive GUI
- `import javax.swing.*;`

Swing



Swing Features

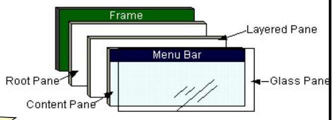
- Swing Components
- Pluggable Look & Feel
- Data Transfer
- Internationalization and Localization
- Accessibility
- **System Tray Icon Support**

Swing Components

- **Swing Components**
  - Basic Controls
    - ✓Jbutton
    - ✓JCheckBox
    - ✓JComboBox
    - ✓Jlist
    - ✓Jmenu
    - ✓JRadioButton
    - ✓Jslider
    - ✓Jspinner
    - ✓JTextField
    - ✓JPasswordField
  - Interactive Displays
    - ✓JColorChooser
    - ✓JEditorPane
    - ✓JFileChooser
    - ✓JTable
    - ✓JTextArea
    - ✓JTree
- Uneditable Information Display
  - ✓JLabel
  - ✓JProgressBar
  - ✓JSeperator
  - ✓JToolTip

Swing Components..

- Top Level Container
  - ✓Jframe
  - ✓JDialog
  - ✓JApplet
- General Purpose Container
  - ✓JPanel
  - ✓JScrollPane
  - ✓JSplitPane
  - ✓JTabbed Pane
  - ✓JToolBar
- Special Purpose Container
  - ✓JInternalFrame
  - ✓JlayeredPane
  - ✓JRootPane



Don't add a component directly to a top-level container.

Top Level Containers (cont)

Pluggable look & Feel

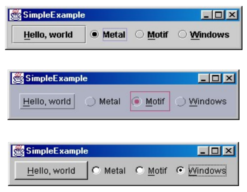
- Swing toolkit allows you to decide how to configure the particular look and feel of your application
- If you don't specify a look and feel, the Swing UI manager figures out which one to use
- Swing ships with four look and feels:
  - Java (also called Metal)
  - Microsoft Windows
  - CDE/Motif
  - GTK+ (GIMP Tool Kit)
    - ✓cross-platform widget toolkit for creating graphical user interfaces)
    - ✓GNU Image Manipulation Program
  - *there are many available for free on the Internet.*

Pluggable look & Feel..

- options for setting a look and feel include
  - Leave it up to the Swing UI manager
  - Use the look and feel of the native platform
  - Specify a particular look and feel
  - Create your own look and feel using the Synth package.
  - Use an externally provided look and feel.

Pluggable Look and Feel

Each picture shows the same program but with a different look and feel



Data Transfer

- Swing toolkit supports the ability to transfer data between components
  - within the same Java application
  - between different Java applications
  - between Java and native applications
- Data can be transferred via
  - a drag and drop gesture
  - or via the clipboard using cut, copy, and paste

Internationalization and Localization

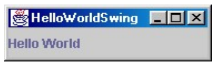
- the process of designing an application so that the user can run it using his or her cultural preferences without modifying or recompiling the code
- cultural preferences, collectively known as *locale*, include :
  - language
  - currency formatting
  - time and date formatting
  - numeric formatting
- *Localization* is the process of translating the text to a particular language and adding any locale-specific components.

Accessibility

- Assistive technologies exist to enable people with permanent or temporary disabilities to use the computer.
- This includes a wide variety of techniques and equipment
  - voice interfaces
  - Magnifiers
  - screen readers
  - closed captioning
  - keyboard enhancements, and so on
- A certain level of accessibility is built-in to all Swing components, but full accessibility can be achieved by following some simple rules
- For example, assign tool tips, keyboard alternatives, and textual descriptions for images, wherever possible.

Example 1

```
import javax.swing.*;

public class HelloWorldSwing {

  public static void main(String[] args) {
    JFrame frame = new JFrame("HelloWorldSwing");
    final JLabel label = new JLabel("Hello World");
    frame.getContentPane().add(label);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
}
```



pack() causes a window to be sized to fit the preferred size and layouts of its sub-components

Example 2

```
import javax.swing.*;

public class HelloWorldFrame extends JFrame
  public HelloWorldFrame() {
    super("HelloWorldSwing");
    final JLabel label = new JLabel("Hello World");
    getContentPane().add(label);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
    setVisible(true);
  }
  public static void main(String[] args) {
    HelloWorldFrame frame = new HelloWorldFrame();
  }
```

In this example a custom frame is created

JDialog

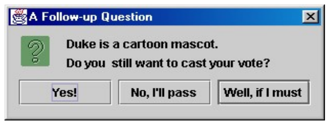
- Every dialog is dependent on a frame
  - Destroying a frame destroys all its dependent dialogs.
  - When the frame is iconified, its dependent dialogs disappear from the screen.
  - When the frame is deiconified, its dependent dialogs return to the screen.
- A dialog can be **modal**. When a modal dialog is visible it blocks user input to all other windows in the program.

JDialog (cont)

- To create custom dialogs, use the JDialog class directly (as in the previous examples).
- Swing provides several standard dialogs
  - JProgressBar, JFileChooser, JColorChooser, ...
- The JOptionPane class can be used to create simple modal dialogs
  - icons, title, text and buttons can be customized.

Example 3

```
Object[] options = {"Yes!", "No, I'll pass",
                    "Well, if I must"};
int n = JOptionPane.showOptionDialog(
        frame, "Duke is a cartoon mascot. \n" +
        "Do you still want to cast your vote?",
        "A Follow-up Question",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null,
        options,
        options[2]);
```


A Follow-up Question

Duke is a cartoon mascot.
Do you still want to cast your vote?

Layout Management

- The process of determining the size and position of components.
- Layout management can be done using **absolute positioning**
  - Size and position of every component within the container must be specified.
  - Does not adjust well when the top-level container is resized.
  - Does not adjust well to differences between users and systems, such as font size.

Layout Management (cont)

- Layout management is often performed using **layout mangers**
  - Components can provide size and position *hints* to layout managers, but layout managers have the final say on the size and position of those components.

Layout Management (cont)

- Layout hints
  - Minimum, preferred and maximum size
  - X axis alignment, Y axis alignment
- Customizing layout hints
  - Invoking setter methods: setMinimumSize, setAlignmentX, ...
  - Subclassing and overriding the getter methods: getMinimumSize, getAlignmentX, ...

Layout Management (cont)

- The Java platform supplies five commonly used layout managers:
  - BorderLayout
  - BoxLayout
  - FlowLayout
  - GridLayout
  - GridBagLayout

Layout Management (cont)

- When using the *add* method to put a component in a container, the container's layout manager must be taken into account.
  - Relative position (BorderLayout)

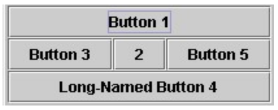
```
panel.add(component, BorderLayout.CENTER);
```

  - Order of addition (BoxLayout, GridLayout, ...)

```
panel.add(component);
```

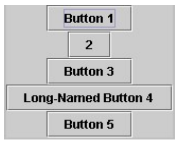
BorderLayout

- Has five areas available to hold components
  - north, south, east, west and center
- All extra space is placed in the center area
  - Only the center area is affected when the container is resized.
- Default layout manager of content panes.



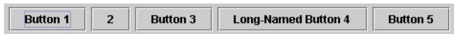
BoxLayout

- Places components in a single row (left to right) or column (top to bottom).
- Respects component's maximum size and alignment hints.



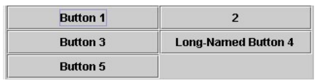
FlowLayout

- Places components from left to right, starting new rows if necessary.
- Default LayoutManager of JPanel



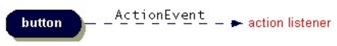
GridLayout

- Places components in a requested number of rows and columns.
- Components are placed left-to-right and top-to-bottom.
- Forces all components to be the same size
  - as wide as the widest component's preferred width
  - as high as the highest component's preferred height



Events Handling

- Every time a user types a character or pushes a mouse button, an ***event*** occurs.
- Any object can be notified of an event by registering as an ***event listener*** on the appropriate ***event source***.
- Multiple listeners can register to be notified of events of a particular type from a particular source.



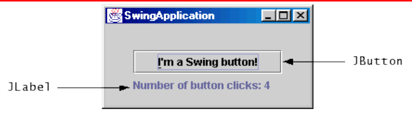
Types of Event Listeners

Act that results in event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

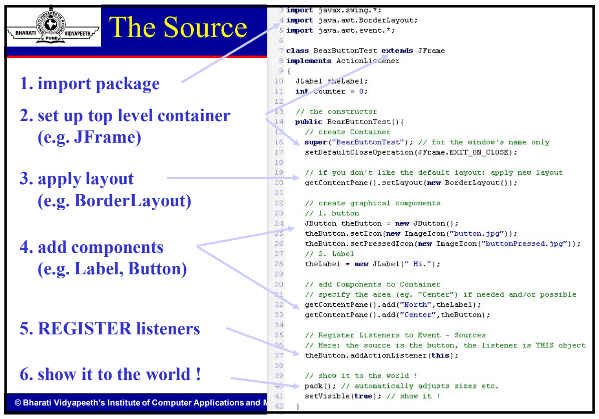
Implementing an Event Handler

- Implement a listener interface or extend a class that implements a listener interface.
- Register an instance of the event handler class as a listener upon one or more components.
- Implement the methods in the listener interface to handle the event.

Example 4



```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
  }});
```

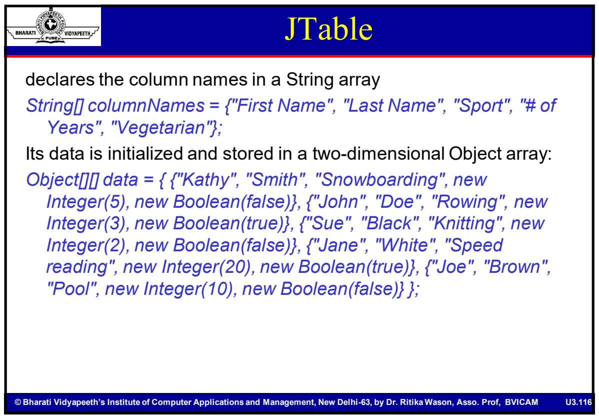


The Source

1. import package
2. set up top level container (e.g. JFrame)
3. apply layout (e.g. BorderLayout)
4. add components (e.g. Label, Button)
5. REGISTER listeners
6. show it to the world !

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

class BearButtonTest extends JFrame
implements ActionListener
{
  JLabel theLabel;
  int counter = 0;

  // the constructor
  public BearButtonTest(){
    // create Container
    super("BearButtonTest"); // for the window's name only
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // if you don't like the default layout: apply new layout
    getContentPane().setLayout(new BorderLayout());

    // create graphical components
    // 1. button
    JButton theButton = new JButton();
    theButton.setIcon(new ImageIcon("button.jpg"));
    theButton.setPressedIcon(new ImageIcon("buttonPressed.jpg"));
    // 2. Label
    theLabel = new JLabel(" Hi.");

    // add Components to Container
    // specify the area (eg. "Center") if needed and/or possible
    getContentPane().add("North",theLabel);
    getContentPane().add("Center",theButton);

    // Register Listeners to Event - Sources
    // Here: the source is the button, the listener is THIS object
    theButton.addActionListener(this);

    // show it to the world !
    pack(); // automatically adjusts sizes etc.
    setVisible(true); // show it !
  }
```

© Bharati Vidyapeeth's Institute of Computer Applications and M



JTable

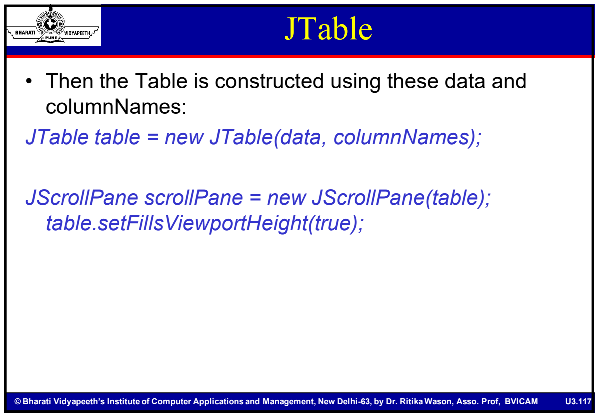
declares the column names in a String array

String[] columnNames = {"First Name", "Last Name", "Sport", "# of Years", "Vegetarian"};

Its data is initialized and stored in a two-dimensional Object array:

Object[][] data = { {"Kathy", "Smith", "Snowboarding", new Integer(5), new Boolean(false)}, {"John", "Doe", "Rowing", new Integer(3), new Boolean(true)}, {"Sue", "Black", "Knitting", new Integer(2), new Boolean(false)}, {"Jane", "White", "Speed reading", new Integer(20), new Boolean(true)}, {"Joe", "Brown", "Pool", new Integer(10), new Boolean(false)} };

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.116



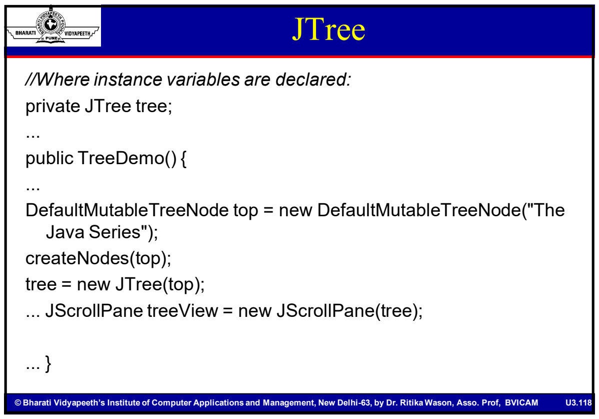
JTable

- Then the Table is constructed using these data and columnNames:

JTable table = new JTable(data, columnNames);

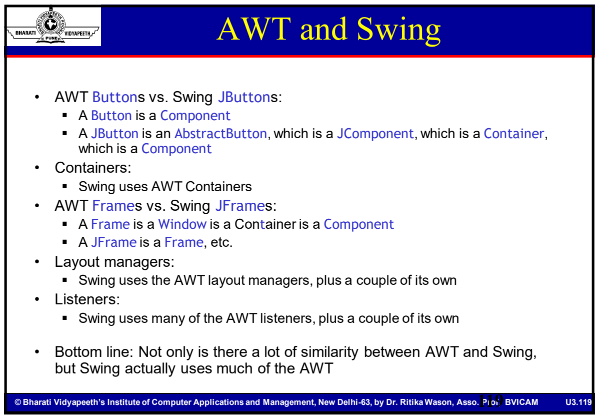
JScrollPane scrollPane = new JScrollPane(table); table.setFillsViewportHeight(true);

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U3.117



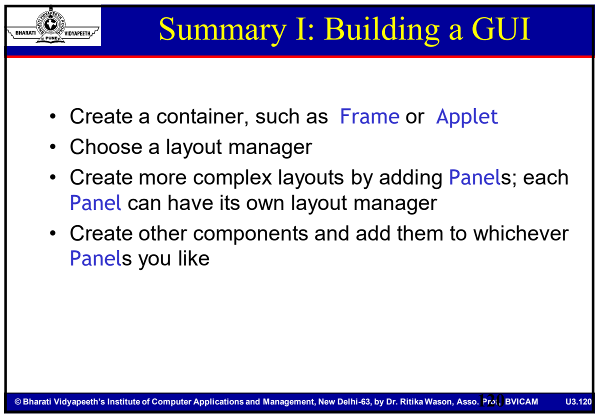
JTree

```
//Where instance variables are declared:
private JTree tree;
...
public TreeDemo() {
...
DefaultMutableTreeNode top = new DefaultMutableTreeNode("The
    Java Series");
createNodes(top);
tree = new JTree(top);
... JScrollPane treeView = new JScrollPane(tree);

... }
```



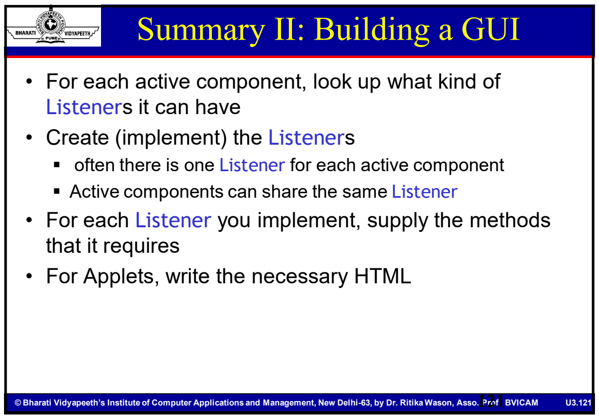
AWT and Swing

- AWT Buttons vs. Swing JButtons:
  - A Button is a Component
  - A JButton is an AbstractButton, which is a JComponent, which is a Container, which is a Component
- Containers:
  - Swing uses AWT Containers
- AWT Frames vs. Swing JFrames:
  - A Frame is a Window is a Container is a Component
  - A JFrame is a Frame, etc.
- Layout managers:
  - Swing uses the AWT layout managers, plus a couple of its own
- Listeners:
  - Swing uses many of the AWT listeners, plus a couple of its own
- Bottom line: Not only is there a lot of similarity between AWT and Swing, but Swing actually uses much of the AWT



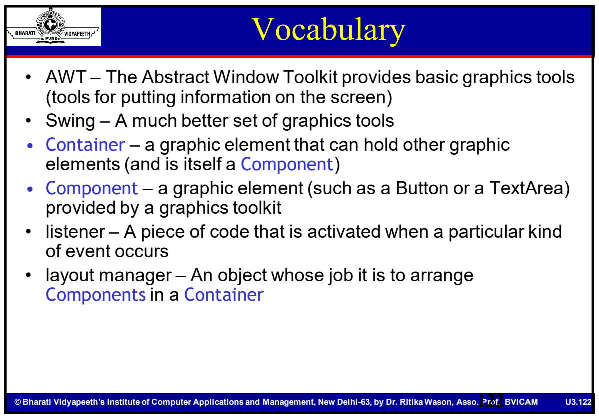
Summary I: Building a GUI

- Create a container, such as Frame or Applet
- Choose a layout manager
- Create more complex layouts by adding Panels; each Panel can have its own layout manager
- Create other components and add them to whichever Panels you like



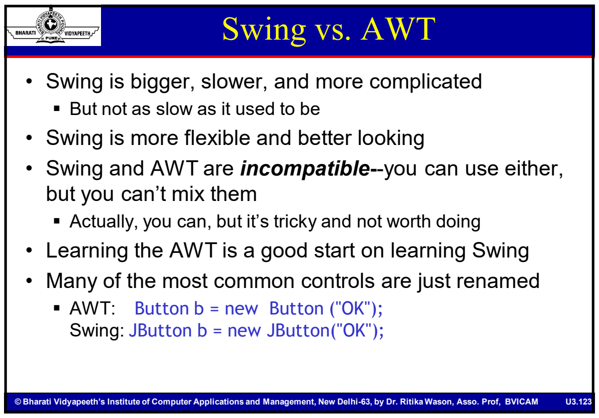
Summary II: Building a GUI

- For each active component, look up what kind of Listeners it can have
- Create (implement) the Listeners
  - often there is one Listener for each active component
  - Active components can share the same Listener
- For each Listener you implement, supply the methods that it requires
- For Applets, write the necessary HTML



Vocabulary

- AWT – The Abstract Window Toolkit provides basic graphics tools (tools for putting information on the screen)
- Swing – A much better set of graphics tools
- Container – a graphic element that can hold other graphic elements (and is itself a Component)
- Component – a graphic element (such as a Button or a TextArea) provided by a graphics toolkit
- listener – A piece of code that is activated when a particular kind of event occurs
- layout manager – An object whose job it is to arrange Components in a Container



Swing vs. AWT

- Swing is bigger, slower, and more complicated
  - But not as slow as it used to be
- Swing is more flexible and better looking
- Swing and AWT are ***incompatible***--you can use either, but you can't mix them
  - Actually, you can, but it's tricky and not worth doing
- Learning the AWT is a good start on learning Swing
- Many of the most common controls are just renamed
  - AWT: `Button b = new  Button ("OK");`
    Swing: `JButton b = new JButton("OK");`