Lecture 39
A W T Programming - I

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

Java AWT Programming – I

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

An applet is ...

- An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. An applet is a Java class that extends the `java.applet.Applet` class. A `main()` method is not invoked on an applet, and an applet class will not define main().

- According to Sun "An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application....The `Applet` class provides a standard interface between applets and their environment."

- Four definitions of applet:
  - A small application.
  - A secure program that runs inside a web browser.
  - A subclass of `java.applet.Applet`
  - An instance of a subclass of `java.applet.Applet`

An applet can...

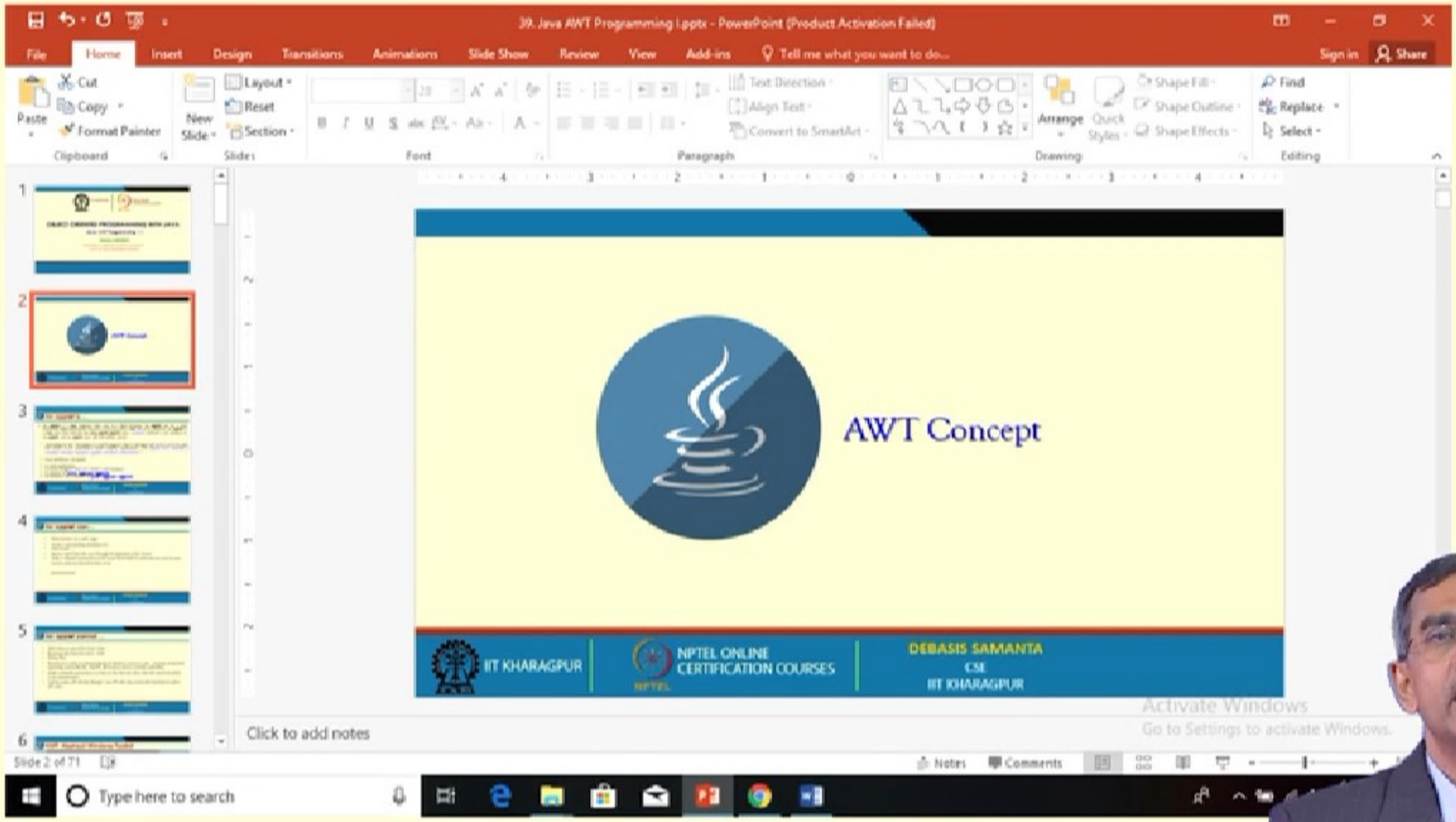
- Draw pictures on a web page.
- Create a new window and draw in it.
- Play sounds, videos.
- Receive input from the user through the keyboard or the mouse.
- Make a network connection to the server from which it came and can send to and receive arbitrary data from that server.

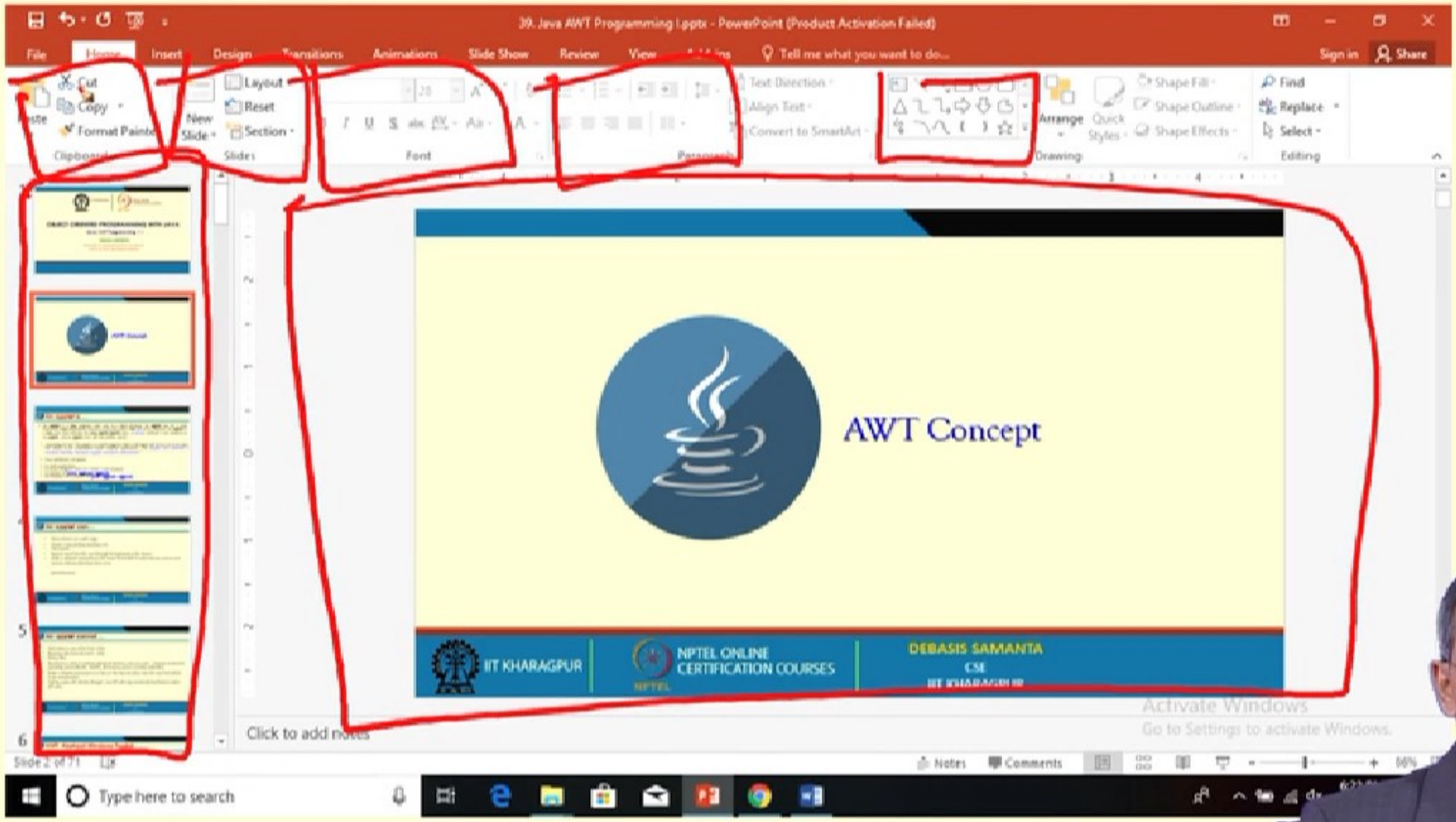
. . . *And many more*

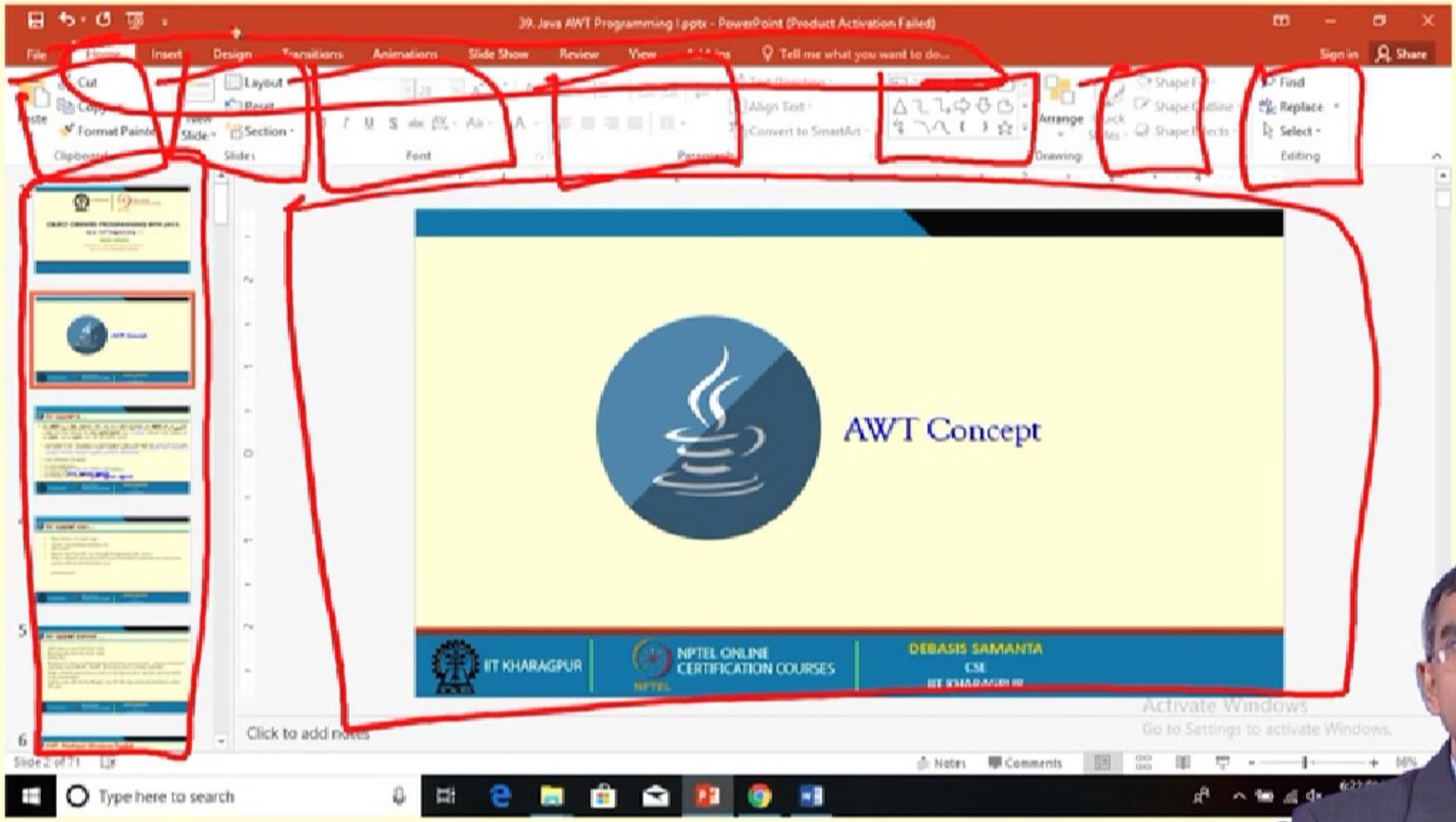
An applet cannot ...

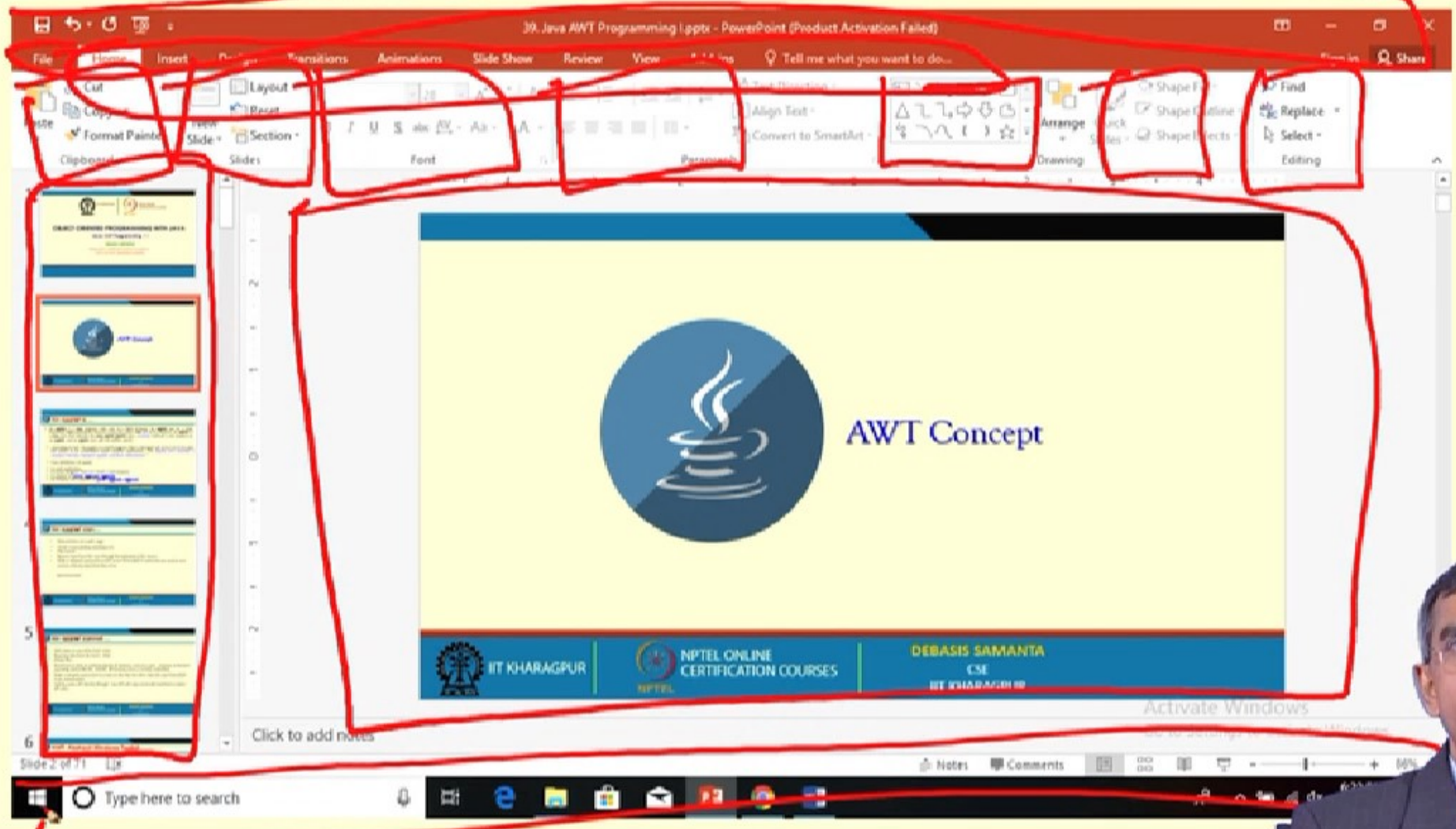
- Write data on any of the host's disks.
- Read any data from the host's disks.
- Delete files.
- Read from or write to arbitrary blocks of memory, even on a non-memory-protected operating system like the MacOS. All memory access is strictly controlled.
- Make a network connection to a host on the Internet other than the one from which it was downloaded.
- Call the native API directly.

AWT Concept

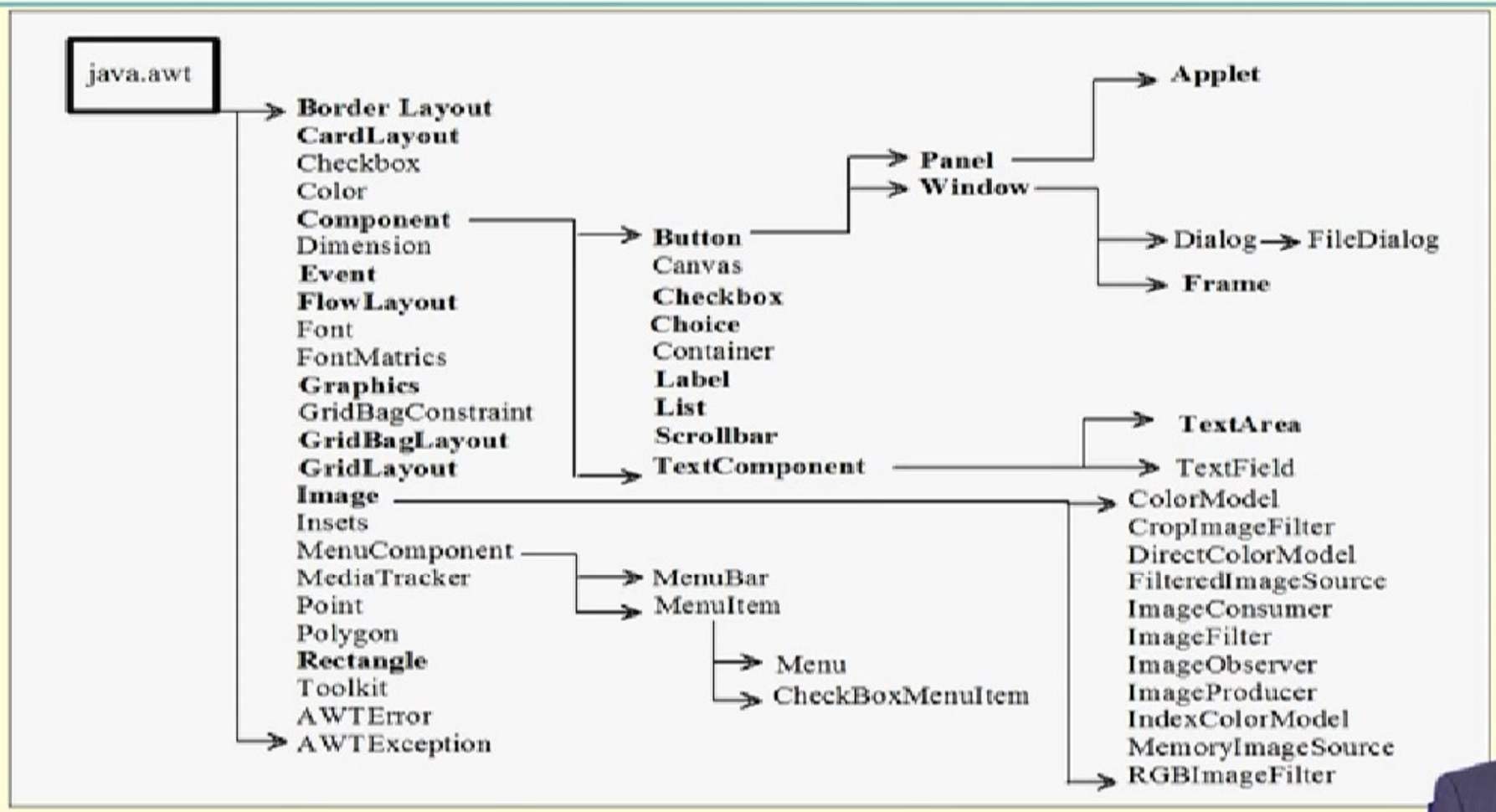

39. Java AWT Programming I.pptx - PowerPoint (Product Activation Failed)
AWT Concept
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE
IIT KHARAGPUR
Click to add notes
Type here to search


39. Java AWT Programming I.pptx - PowerPoint (Product Activation Failed)
AWT Concept
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE
Click to add notes
Type here to search


39. Java AWT Programming I.pptx - PowerPoint (Product Activation Failed)
AWT Concept
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE
Click to add notes
Type here to search

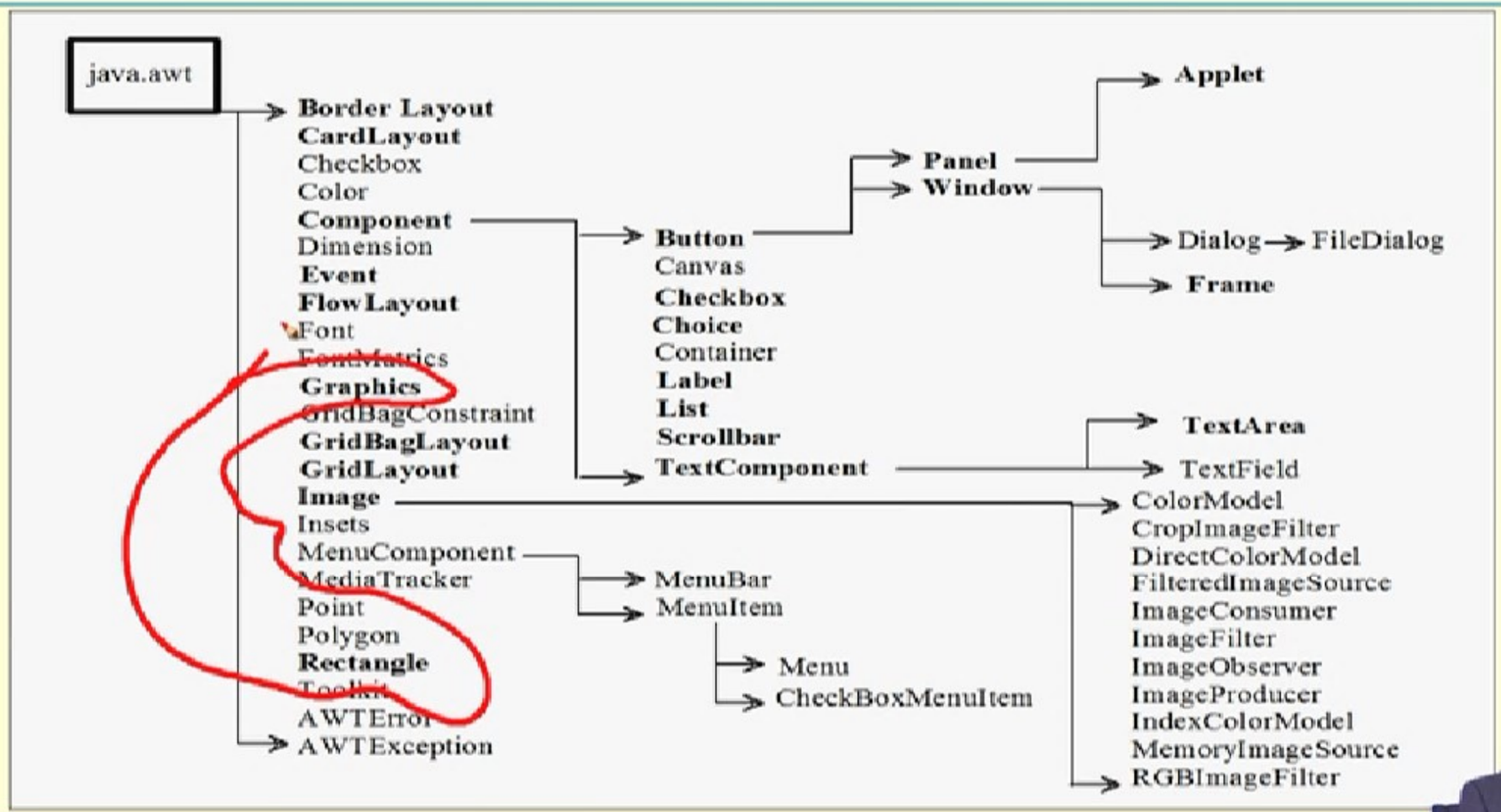

39. Java AWT Programming I.pptx - PowerPoint (Product Activation Failed)
AWT Concept
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE
Click to add notes
Type here to search

AWT: Abstract Windows Toolkit



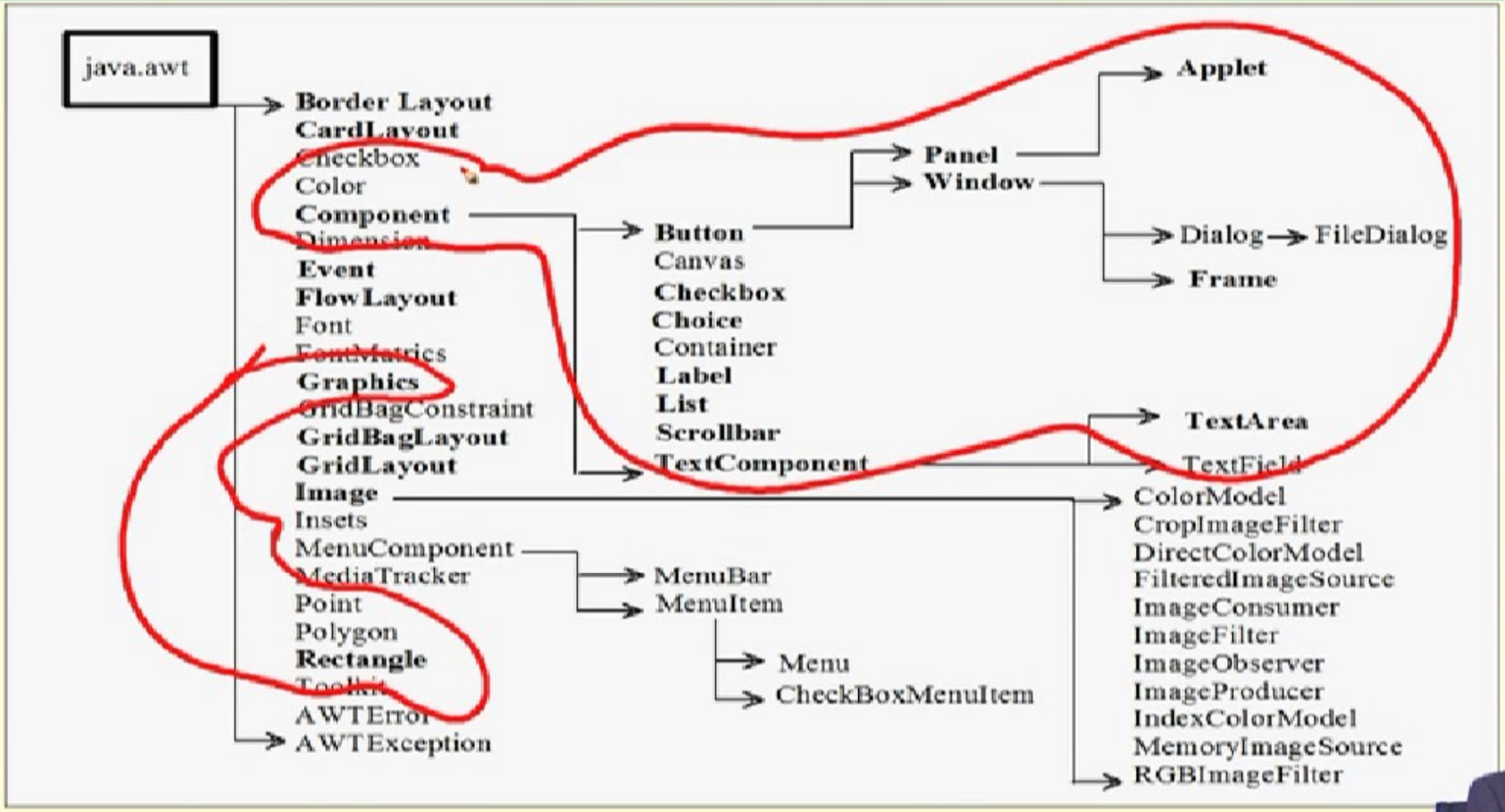
5 Sec. Backward - [00:12:00 / 31%]

AWT: Abstract Windows Toolkit

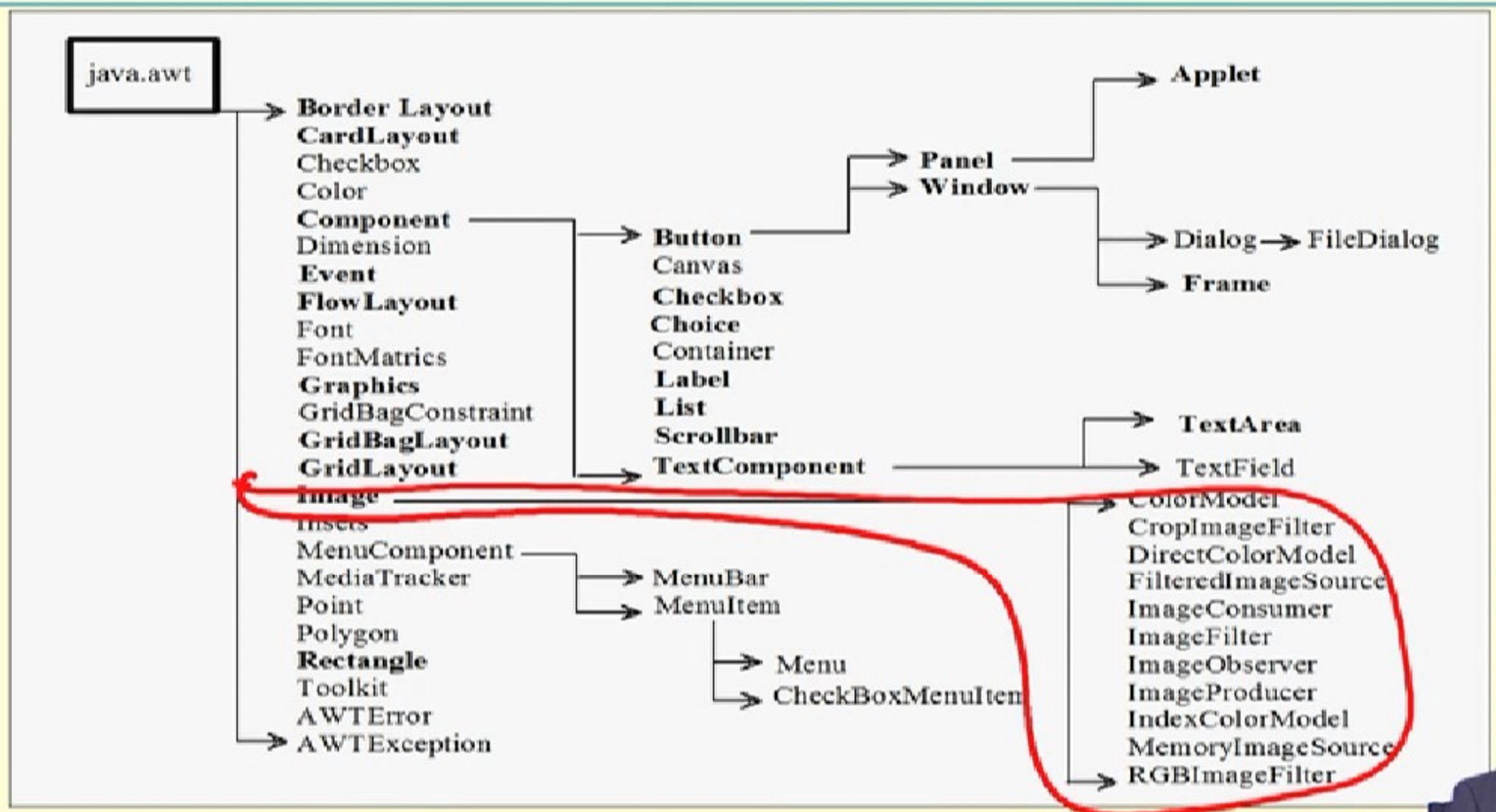


5 Sec. Forward - [00:12:12 / 32%]

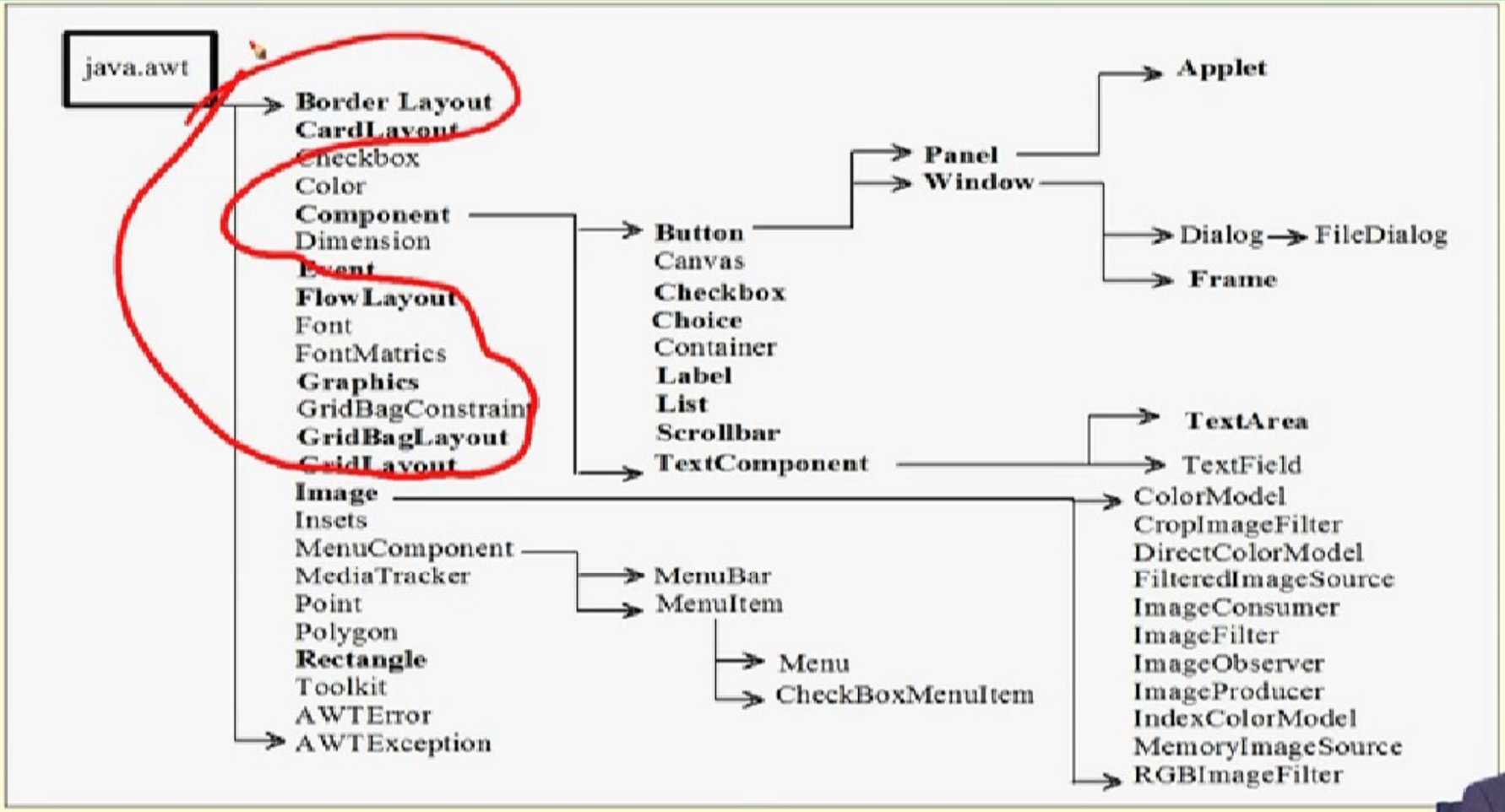
AWT: Abstract Windows Toolkit



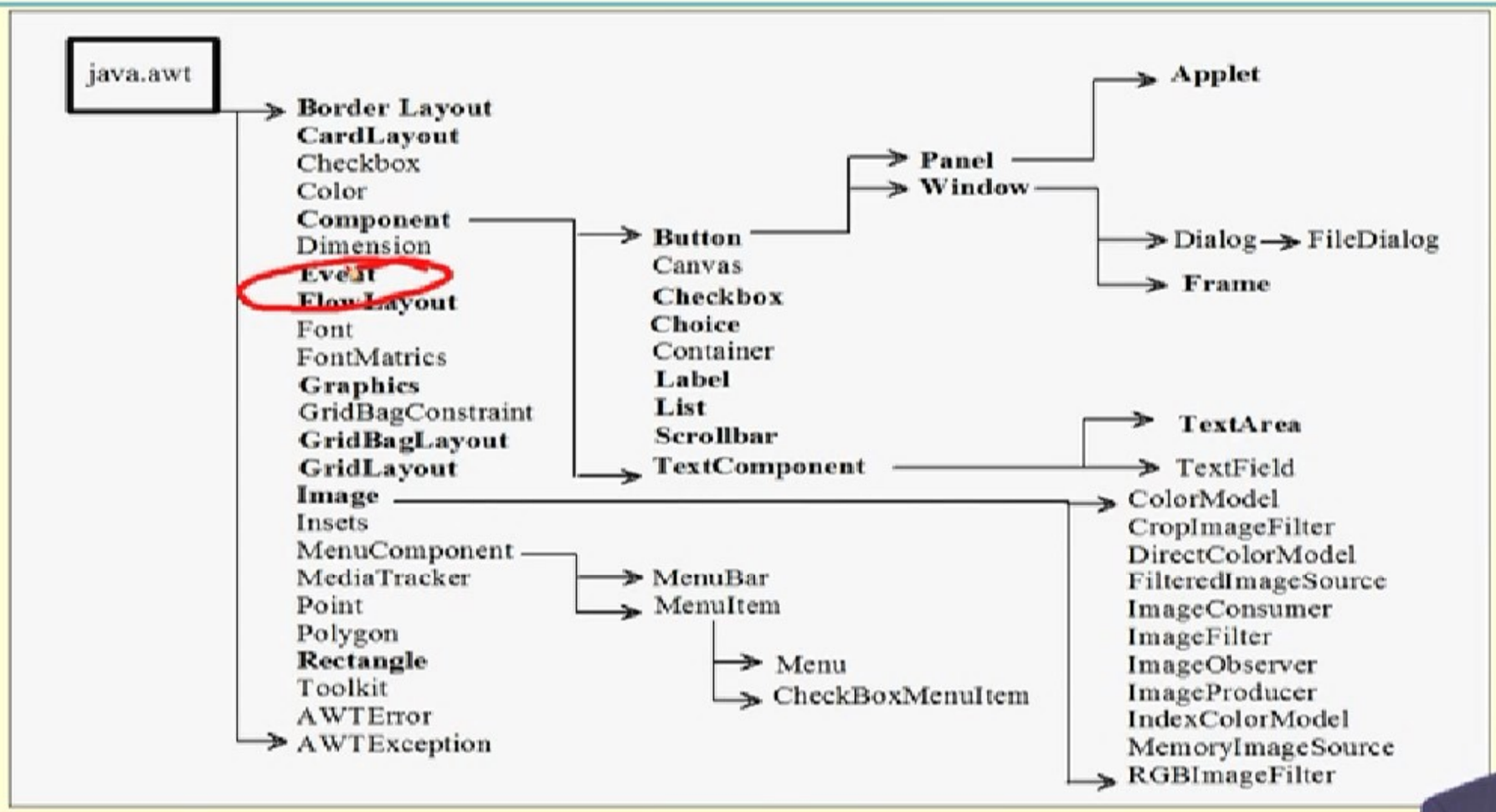
AWT: Abstract Windows Toolkit



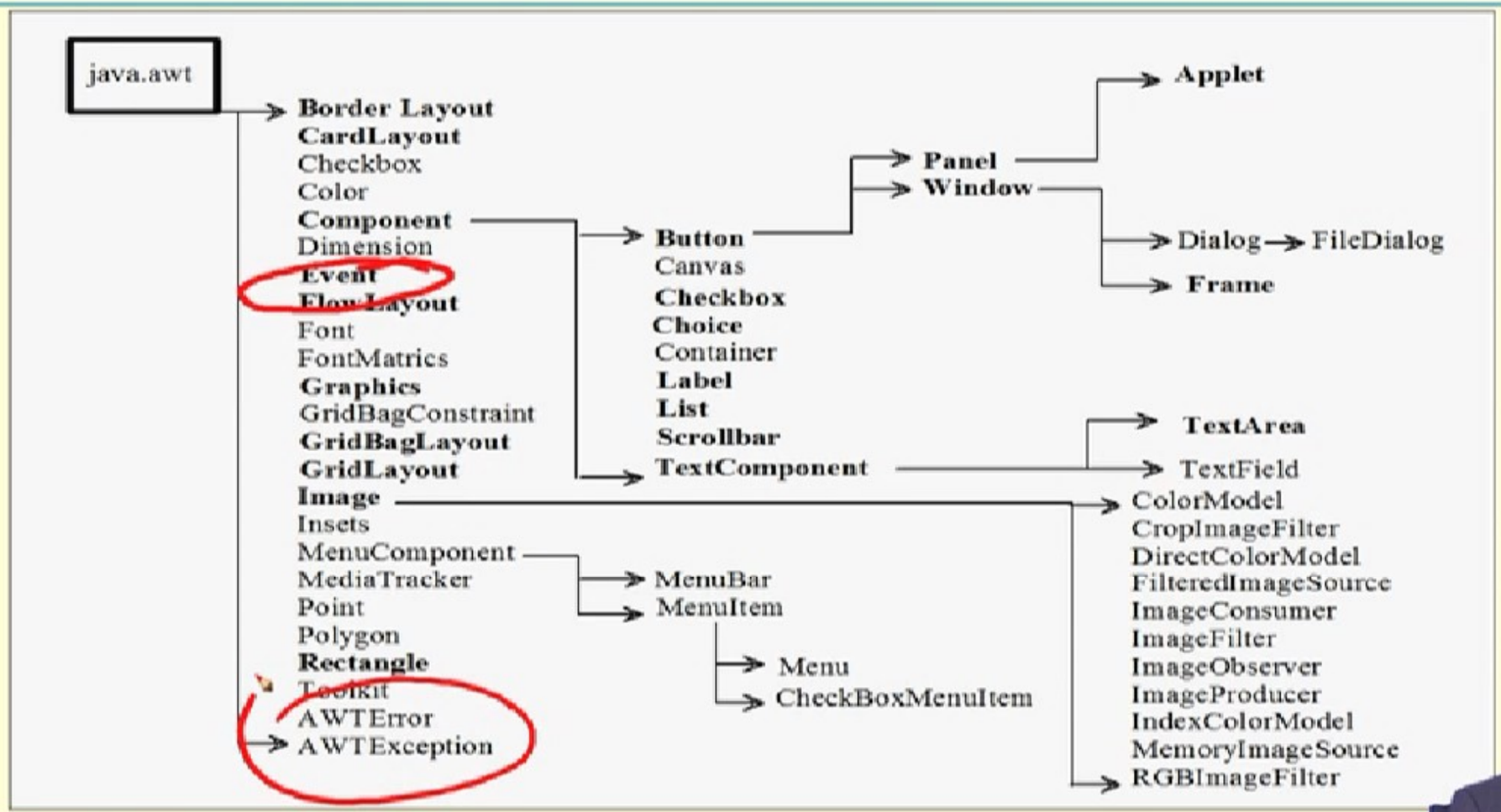
AWT: Abstract Windows Toolkit



AWT: Abstract Windows Toolkit



AWT: Abstract Windows Toolkit



Applet and AWT

- Abstract Window Toolkit (**AWT**) is a set of application program interfaces (APIs) used by **Java** programmers to create graphical user interface (GUI) objects, such as buttons, scroll bars, and windows.
- **AWT** is part of the **Java Developer's Kit (JDK)** from Sun Microsystems, the company that originated **Java**.

Applet and AWT

- Abstract Window Toolkit (**AWT**) is a set of application program interfaces (APIs) used by **Java** programmers to create graphical user interface (GUI) objects, such as buttons, scroll bars, and windows.
- **AWT** is part of the **Java Developer's Kit (JDK)** from Sun Microsystems, the company that originated **Java.**

```
public class Applet extends Panel
```

```
java.lang.Object
        |
        +----java.awt.Component
                    |
                    +----java.awt.Container
                                |
                                +----java.awt.Panel
                                            |
                                            +----java.applet.Applet
```

AWT GUIs

Classes for AWT GUIs

Class	Description
Button	This class creates a labeled button.
Canvas	A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.
CardLayout	A CardLayout object is a layout manager for a container.
Checkbox	A check box is a graphical component that can be in either an "on" (true) or "off" (false) state.
CheckboxGroup	The CheckboxGroup class is used to group together a set of Checkbox buttons.
CheckboxMenuItem	This class represents a check box that can be included in a menu.
Choice	The Choice class presents a pop-up menu of choices.
Color	The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.
Component	A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user.

Classes for AWT GUIs

Class	Description
Dialog	A Dialog is a top-level window with a title and a border that is typically used to take some form of input from the user.
Label	A Label object is a component for placing text in a container.
List	The List component presents the user with a scrolling list of text items.
Menu	A Menu object is a pull-down menu component that is deployed from a menu bar.
MenuBar	The MenuBar class encapsulates the platform's concept of a menu bar bound to a frame.
MenuComponent	The abstract class MenuComponent is the superclass of all menu-related components.
MenuItem	All items in a menu must belong to the class MenuItem, or one of its subclasses.
Panel	Panel is the simplest container class.
TextArea	A TextArea object is a multi-line region that displays text.
TextField	A TextField object is a text component that allows for the editing of a single line of text.

Classes for AWT GUIs

Interface	Description
ActiveEvent	An interface for events that know how to dispatch themselves.
Adjustable	The interface for objects which have an adjustable numeric value contained within a bounded range of values.
Composite	The Composite interface, along with CompositeContext, defines the methods to compose a draw primitive with the underlying graphics area.
CompositeContext	The CompositeContext interface defines the encapsulated and optimized environment for a compositing operation.
ItemSelectable	The interface for objects which contain a set of items for which zero or more can be selected.
KeyEventDispatcher	A KeyEventDispatcher cooperates with the current KeyboardFocusManager in the targeting and dispatching of all KeyEvents.
KeyEventPostProcessor	A KeyEventPostProcessor cooperates with the current KeyboardFocusManager in the final resolution of all unconsumed KeyEvents.
LayoutManager	Defines the interface for classes that know how to lay out Containers.
LayoutManager2	Defines an interface for classes that know how to layout Containers based on a layout constraints object.

Classes for AWT GUIs

Interface	Description
MenuContainer	The super class of all menu related containers.
Paint	This Paint interface defines how color patterns can be generated for Graphics2D operations.
PaintContext	The PaintContext interface defines the encapsulated and optimized environment to generate color patterns in device space for fill or stroke operations on a Graphics2D.
PrintGraphics	An abstract class which provides a print graphics context for a page.
SecondaryLoop	A helper interface to run the nested event loop.
Shape	The Shape interface provides definitions for objects that represent some form of geometric shape.
Stroke	The Stroke interface allows a Graphics2D object to obtain a Shape that is the decorated outline, or stylistic representation of the outline, of the specified Shape.
Transparency	The Transparency interface defines the common transparency modes for implementing classes.

GUI with Components

GUI with Components

- Components are a group of classes which belong to the class Component.
- The basic and frequently used components are

Button	**Checkbox**	**Choice**	**Frame**	**Panel**	**Label**
List	**Scrollbar**	**TextArea**		**TextField**	

- For each of the above components, there are corresponding classes.
- Using a class, an object of desired type can be created.
- Each class have their own constructors for initialization of the objects.
- When an object is created and initialized, it then can be placed on an applet by the `add()` method, which is defined in their corresponding classes.

Methods in Component classes

Method	*Description*
`public void add(Component c)`	Inserts a component on this component.
`public void setSize(int width,int height)`	Sets the size (width and height) of the component.
`public void setLayout(LayoutManager m)`	Defines the layout manager for the component.
`public void setVisible(boolean status)`	Changes the visibility of the component, by default false.

Creating a Frame

A Frame

```
public class Frame extends Window implements MenuContainer
```

A **Frame** is a top-level window with a title and a border.

Class Frame : Constructors

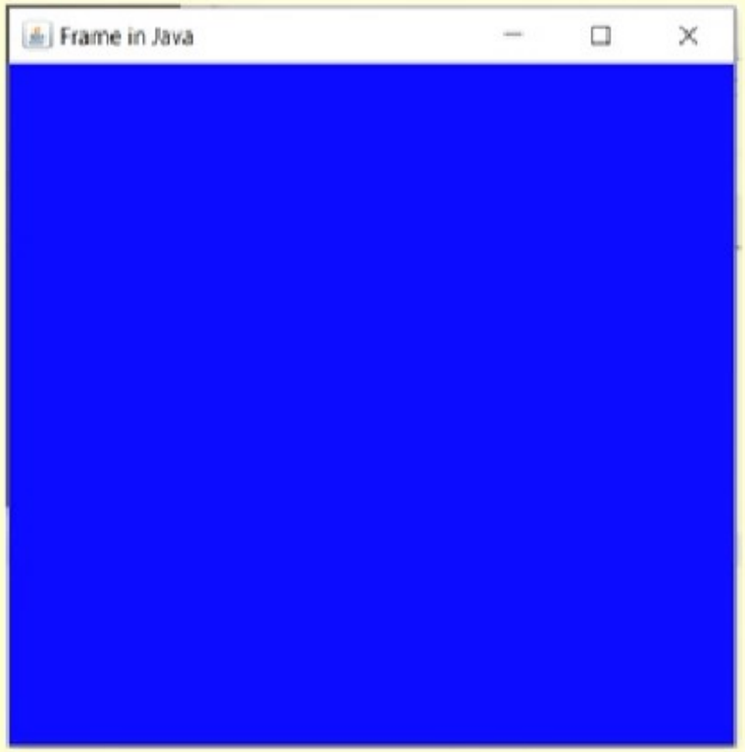
Constructor	Description
Frame ()	Constructs a new instance of Frame that is initially invisible.
Frame (GraphicsConfiguration gc)	Constructs a new, initially invisible Frame with the specified GraphicsConfiguration.
Frame (String title)	Constructs a new, initially invisible Frame object with the specified title.
Frame(String title, GraphicsConfiguration gc)	Constructs a new, initially invisible Frame object with the specified title and a GraphicsConfiguration.

Class Frame : Methods

Modifier and Type	*Method*	*Description*
void	addNotify()	Makes this Frame displayable by connecting it to a native screen resource.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Frame.
int	getExtendedState()	Gets the state of this frame.
static Frame[]	getFrames()	Returns an array of all Frames created by this application.
Image	getIconImage()	Returns the image to be displayed as the icon for this frame.
Rectangle	getMaximizedBounds()	Gets maximized bounds for this frame.
MenuBar	getMenuBar()	Gets the menu bar for this frame.
int	getState()	Gets the state of this frame (obsolete).
String	getTitle()	Gets the title of the frame.
boolean	isResizable()	
boolean	isUndecorated()	Indicates whether this frame is undecorated.
protected String	paramString()	Returns a string representing the state of this Frame.
void	remove(MenuComponent m)	Removes the specified menu bar from this frame.
void	removeNotify()	Makes this Frame undisplayable by removing its connection to its native screen resource.
void	setBackground(Color bgColor)	Sets the background color of this window.

Creating a Frame : An example

```
import java.awt.*;

public class MyFrame {
    public static void main (String args[ ] ) {
            Frame frame =  new Frame ( "Frame in Java " );
            frame.resize (500, 500);
            frame.setBackground (Color.blue);

            frame.show ( );
    }
}
```

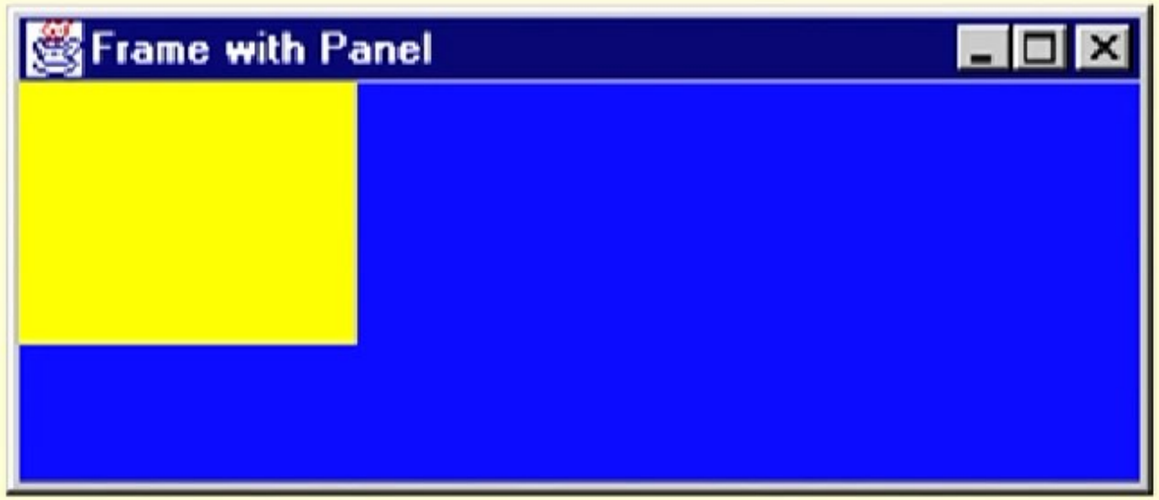


Creating a Panel

A Panel

```
public class Panel extends Container implements Accessible
```

The **Panel** is a simplest container class. It provides space in which an application can attach any other component. It inherits the **Container** class. It doesn't have title bar.



Class Panel : Constructors

Constructor	Description
`Panel()`	Creates a new panel using the default layout manager.
`Panel(LayoutManager layout)`	Creates a new panel with the specified layout manager.

Class Panel : Methods

Modifier and Type	*Method*	*Description*
void	addNotify()	Creates the Panel's peer.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Panel.

Creating a Panel : An example

```
import java.awt.* ;

public class MyPanel  {
    public static void main ( String args [ ] )  {
            Frame frame = new Frame( "Frame with panel");
            Panel panel = new Panel(  );
            frame.resize(200, 200);
            frame.setBackground(Color.blue);
            frame.setLayout (null);
            panel.resize (100, 100) ;
            panel.setBackground (Color.yellow );
            frame.add (panel);
            frame.show ( );
    }
}
```

Creating a Button

A Button

```
public class Button extends Component implements Accessible
```

This class creates a labeled button. The application can cause some action to happen when the button is pushed. This image depicts three views of a "Quit" button.

- The first view shows the button as it appears normally. The second view shows the button when it has input focus. Its outline is darkened to let the user know that it is an active object. The third view shows the button when the user clicks the mouse over the button, and thus requests that an action be performed.

- If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

A Button

```
public class Button extends Component implements Accessible
```

This class creates a labeled button. The application can cause some action to happen when the button is pushed. This image depicts three views of a "Quit" button.

- The first view shows the button as it appears normally. The second view shows the button when it has input focus. Its outline is darkened to let the user know that it is an active object. The third view shows the button when the user clicks the mouse over the button, and thus requests that an action be performed.

- If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

Class Button : Constructors

Constructor	Description
`Button()`	Constructs a button with an empty string for its label.
`Button(String label)`	Constructs a button with the specified label.

Class Button : Methods

Modifier and Type	*Method*	*Description*
void	addActionListener(ActionListener l)	Adds the specified action listener to receive action events from this button.
void	addNotify()	Creates the peer of the button.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Button.
String	getActionCommand()	Returns the command name of the action event fired by this button.
ActionListener[]	getActionListeners()	Returns an array of all the action listeners registered on this button.
String	getLabel()	Gets the label of this button.
<T extends EventListener> T[]	getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as *Foo*Listeners upon this Button.
protected String	paramString()	Returns a string representing the state of this Button.
protected void	processActionEvent(ActionEvent e)	Processes action events occurring on this button by dispatching them to any registeredActionListener objects.
protected void	processEvent(AWTEvent e)	Processes events on this button.
void	removeActionListener(ActionListener l)	Removes the specified action listener so that it no longer receives action events from this button.
void	setActionCommand(String command)	Sets the command name for the action event fired by this button.
void	setLabel(String label)	Sets the button's label to be the specified string.

Creating a Button : An example

```
import java.awt.*;
class ButtonDemo extends Frame{
   ButtonDemo(){
        Button b = new Button("Click");
        b.setBounds(30,100,80,30);          // setting button position
        add(b);                              //adding button into frame
        setSize(300,300);          //frame size 300 width and 300 height
        setLayout(null);                                  //No layout manager
        setVisible(true);                    // Make the frame visible
   }
   public static void main(String args[]){
      ButtonDemo b = new ButtonDemo();
   }
}
```

The `setBounds(int xaxis, int yaxis, int width, int height)` method is used in the above example that sets the position of the AWT button.

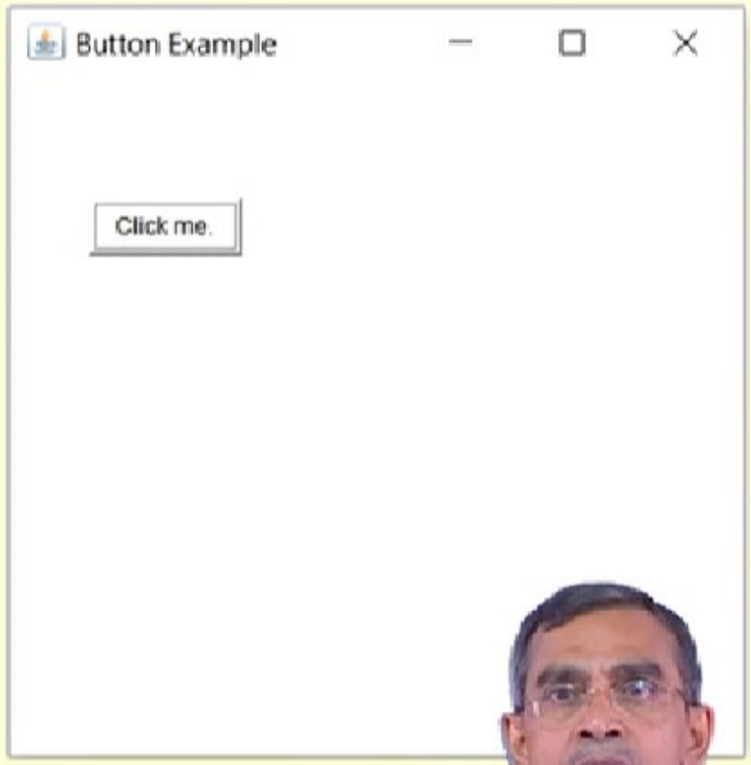
Creating a Button : An example

```
import java.awt.*;
class ButtonDemo extends Frame{
    ButtonDemo(){
        Button b = new Button("Click");
        b.setBounds(30,100,80,30);          // setting button position
        add(b);                             //adding button into frame
        setSize(300,300);          //frame size 300 width and 300 height
        setLayout(null);                                //No layout manager
        setVisible(true);                   // Make the frame visible
    }
    public static void main(String args[]){
        ButtonDemo b = new ButtonDemo();
    }
}
```

The **setBounds(int xaxis, int yaxis, int width, int height)** method is used in the above example that sets the position of the AWT button.

Creating a Button : Another example

```
import java.awt.*;

public class ButtonExample extends Component{
    public static void main(String[] args) {
        Frame f = new Frame("Button Example");
        Button b = new Button("Click me.");
        b.setBounds(50,100,80,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



Creating a Button : One more example

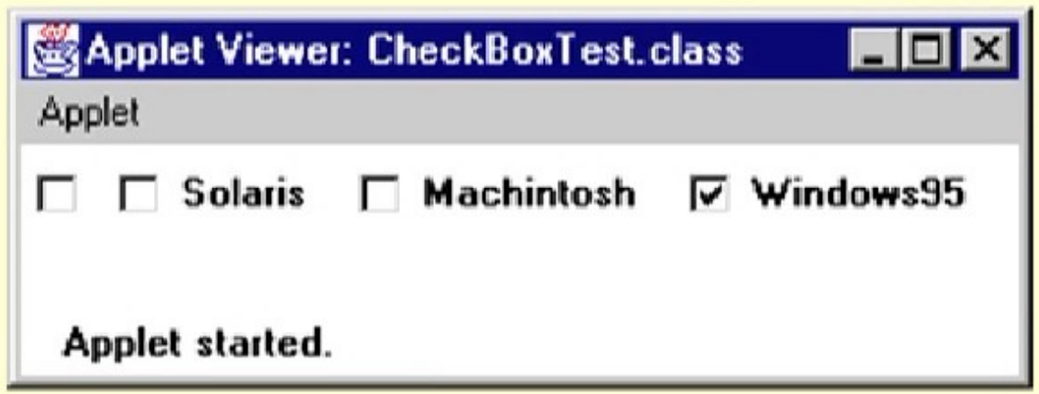
```
import java.applet.Applet;
import java.awt.*;

  public class ButtonTest extends Applet  {
     public void init( ) {
        Button  b1,b2;
        b1 = new Button ("Welcome");
        add(b1);
        b2 = new Button (" ");
        add(b2);
    }
}
```

Creating a Checkbox

A Checkbox

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

A **Checkbox** is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."



Class Checkbox : Constructors

Constructor	Description
Checkbox()	Creates a check box with an empty string for its label.
Checkbox(String label)	Creates a check box with the specified label.
Checkbox(String label, boolean state)	Creates a check box with the specified label and sets the specified state.
Checkbox(String label, boolean state, CheckboxGroup group)	Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
Checkbox(String label, CheckboxGroup group, boolean state)	Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Class Checkbox : Methods

Modifier and Type	*Method*	*Description*
void	add(String item)	Adds an item to this Choice menu.
void	addItem(String item)	Obsolete as of Java 2 platform v1.1.
void	addItemListener(ItemListener l)	Adds the specified item listener to receive item events from this Choice menu.
void	addNotify()	Creates the Choice's peer.
int	countItems()	**Deprecated.** *As of JDK version 1.1, replaced by getItemCount().*
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Choice.
String	getItem(int index)	Gets the string at the specified index in this Choice menu.
int	getItemCount()	Returns the number of items in this Choice menu.
ItemListener[]	getItemListeners()	Returns an array of all the item listeners registered on this choice.
<T extends EventListener> T[]	getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as *Foo*Listeners upon this Choice.
int	getSelectedIndex()	Returns the index of the currently selected item.
String	getSelectedItem()	Gets a representation of the current choice as a string.

Class Checkbox : Methods

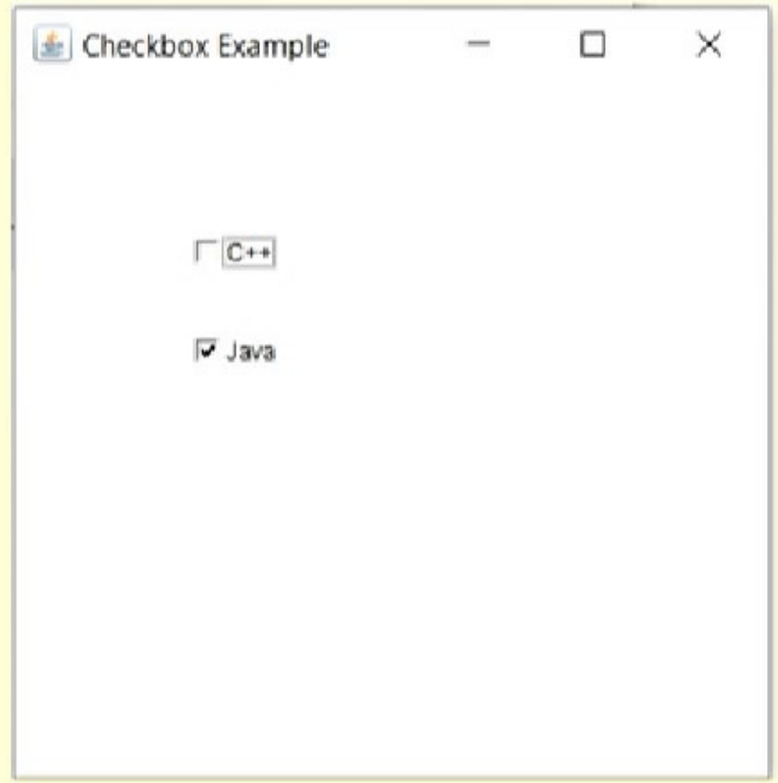
Modifier and Type	*Method*	*Description*
Object[]	getSelectedObjects()	Returns an array (length 1) containing the currently selected item.
void	insert(String item, int index)	Inserts the item into this choice at the specified position.
protected String	paramString()	Returns a string representing the state of this Choice menu.
protected void	processEvent(AWTEvent e)	Processes events on this choice.
protected void	processItemEvent(ItemEvent e)	Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
void	remove(int position)	Removes an item from the choice menu at the specified position.
void	remove(String item)	Removes the first occurrence of item from the Choice menu.
void	removeAll()	Removes all items from the choice menu.
void	removeItemListener(ItemListener l)	Removes the specified item listener so that it no longer receives item events from this Choice menu.
void	select(int pos)	Sets the selected item in this Choice menu to be the item at the specified position.
void	select(String str)	Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

Class Checkbox : Methods

Modifier and Type	*Method*	*Description*
Object[]	getSelectedObjects()	Returns an array (length 1) containing the currently selected item.
void	insert(String item, int index)	Inserts the item into this choice at the specified position.
protected String	paramString()	Returns a string representing the state of this Choice menu.
protected void	processEvent(AWTEvent e)	Processes events on this choice.
protected void	processItemEvent(ItemEvent e)	Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
void	remove(int position)	Removes an item from the choice menu at the specified position.
void	remove(String item)	Removes the first occurrence of item from the Choice menu.
void	removeAll()	Removes all items from the choice menu.
void	removeItemListener(ItemListener l)	Removes the specified item listener so that it no longer receives item events from this Choice menu.
void	select(int pos)	Sets the selected item in this Choice menu to be the item at the specified position.
void	select(String str)	Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

Creating a Checkbox : An example

```
import java.awt.*;

public class CheckboxExample{
  CheckboxExample(){
        Frame f = new Frame("Checkbox Example");
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox checkbox2 = new Checkbox("Java", true);
        checkbox2.setBounds(100,150, 50,50);
        f.add(checkbox1);
        f.add(checkbox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
 }
 public static void main(String args[]){
  new CheckboxExample();
 }
}
```

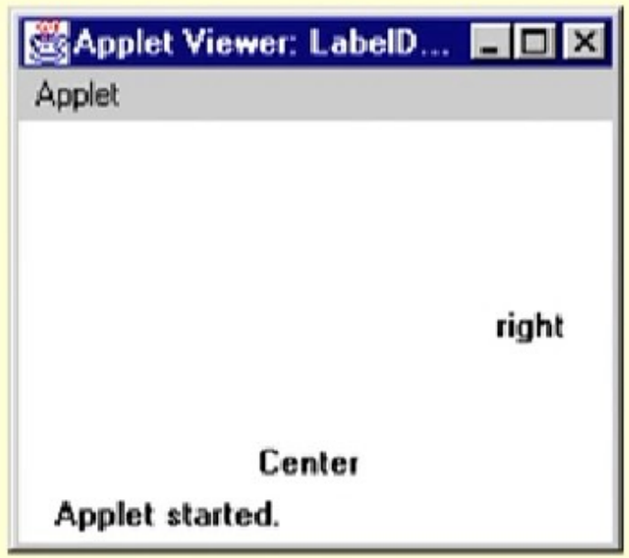


Creating a Label

A Label

```
public class Label extends Component implements Accessible
```

A **Label** object is a component for placing text in a container. A label displays a single line of read-only text. The text can be changed by the application, but a user cannot edit it directly.

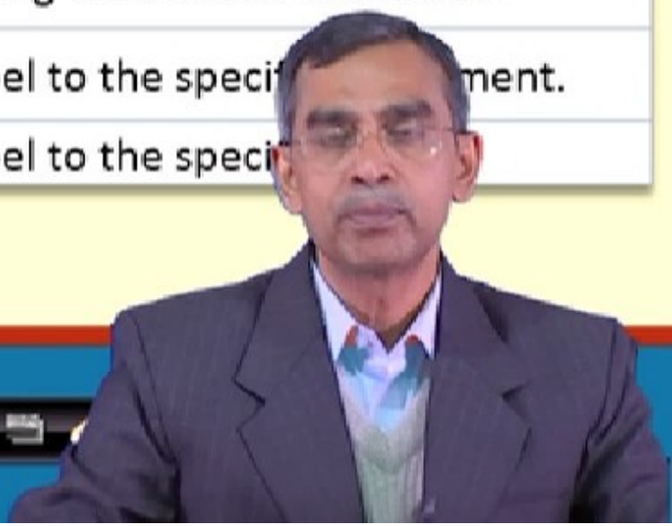


Class Label : Constructors

Constructor	Description
Label()	Constructs an empty label.
Label(String text)	Constructs a new label with the specified string of text, left justified.
Label(String text, int alignment)	Constructs a new label that presents the specified string of text with the specified alignment.

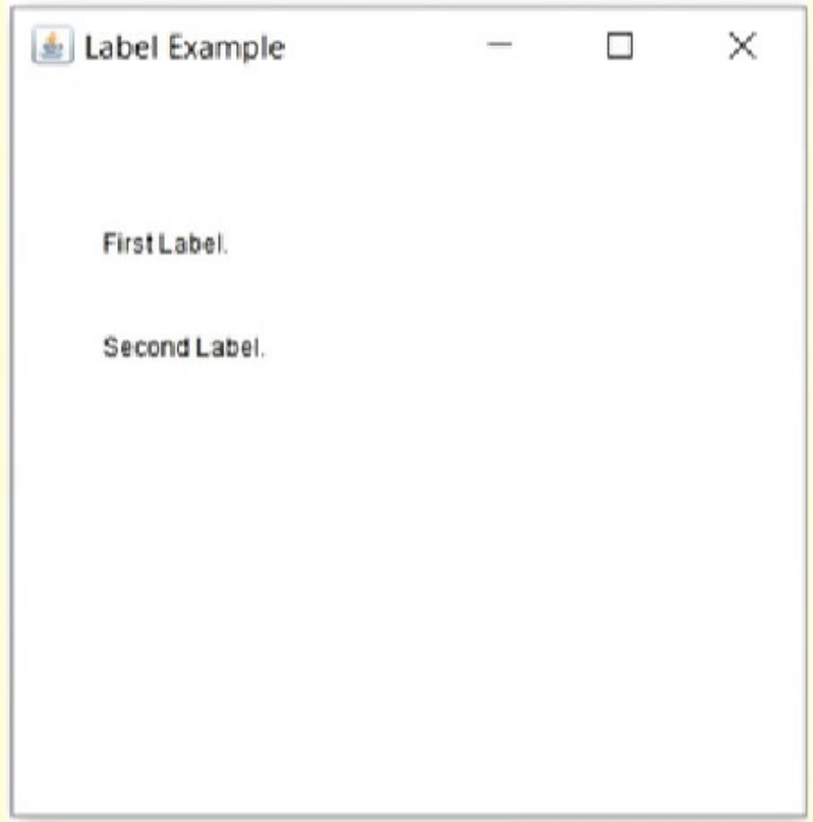
Class Label : Methods

Modifier and Type	*Method*	*Description*
void	addNotify()	Creates the peer for this label.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Label.
int	getAlignment()	Gets the current alignment of this label.
String	getText()	Gets the text of this label.
protected String	paramString()	Returns a string representing the state of this Label.
void	setAlignment(int alignment)	Sets the alignment for this label to the specif ment.
void	setText(String text)	Sets the text for this label to the speci



Creating a Label : An example

```
import java.awt.*;

class MyLabel{
     public static void main(String args[]){
          Frame f = new Frame("Label Example");
          Label l1,l2;
          l1 = new Label("First Label.");
          l1.setBounds(50,100, 100,30);
          l2 = new Label("Second Label.");
          l2.setBounds(50,150, 100,30);
          f.add(l1); f.add(l2);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
     }
}
```

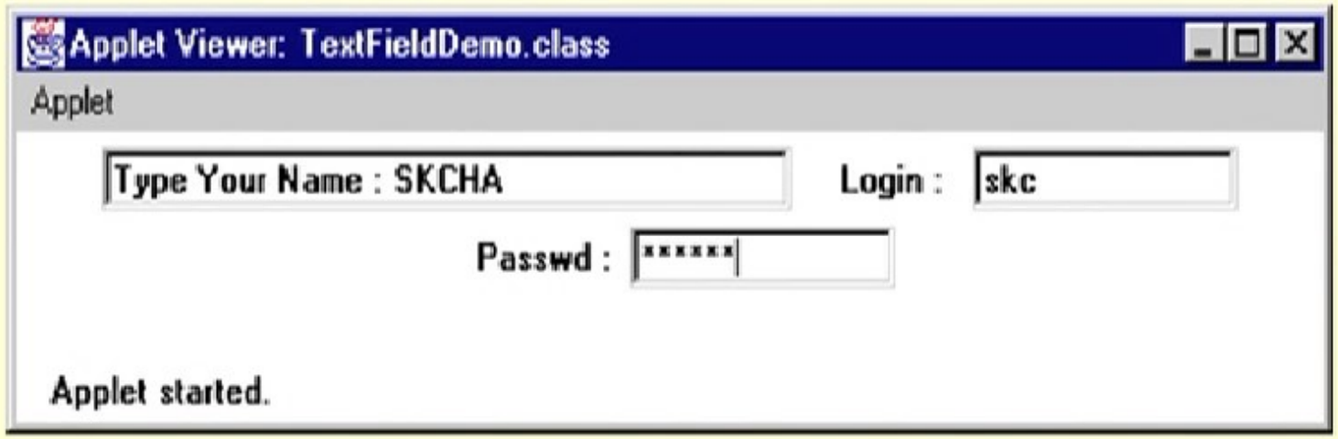


Creating a TextField

A TextField

```
public class TextField extends TextComponent
```

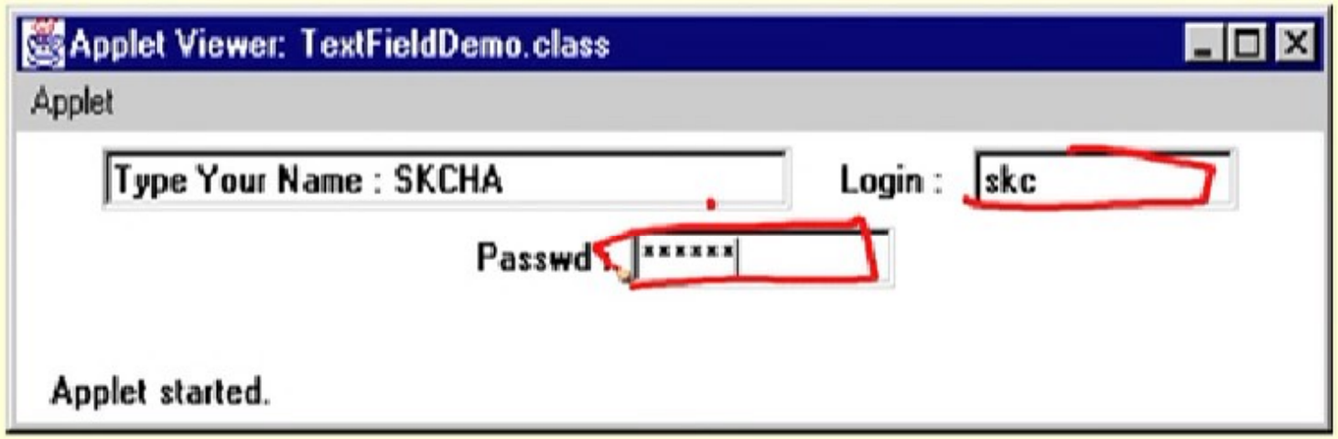
The object of a **TextField** class is a text component that allows the editing of a single line text. It inherits **TextComponent** class.



A TextField

```
public class TextField extends TextComponent
```

The object of a **TextField** class is a text component that allows the editing of a single line text. It inherits **TextComponent** class.

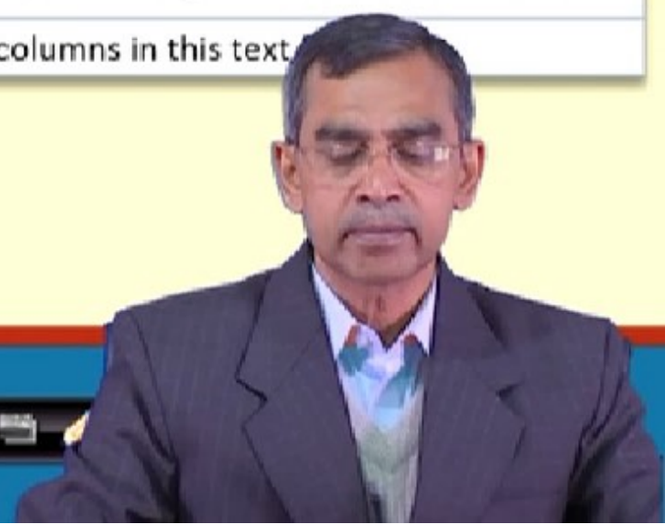


Class TextField : Constructors

Constructor	Description
TextField()	Constructs a new text field.
TextField(int columns)	Constructs a new empty text field with the specified number of columns.
TextField(String text)	Constructs a new text field initialized with the specified text.

Class TextField : Methods

Modifier and Type	*Method*	*Description*
void	addActionListener(ActionListener l)	Adds the specified action listener to receive action events from this text field.
void	addNotify()	Creates the TextField's peer.
boolean	echoCharIsSet()	Indicates whether or not this text field has a character set for echoing.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this TextField.
ActionListener[]	getActionListeners()	Returns an array of all the action listeners registered on this textfield.
int	getColumns()	Gets the number of columns in this text



Class TextField : Methods

Modifier and Type	*Method*	*Description*
Dimension	getMinimumSize()	Gets the minumum dimensions for this text field.
Dimension	getMinimumSize(int columns)	Gets the minumum dimensions for a text field with the specified number of columns.
Dimension	getPreferredSize()	Gets the preferred size of this text field.
Dimension	getPreferredSize(int columns)	Gets the preferred size of this text field with the specified number of columns.
void	setText(String t)	Sets the text that is presented by this text component to be the specified text.
void	setEchoChar(char c)	Sets the echo character for this text field.

Creating a Textfield : An example

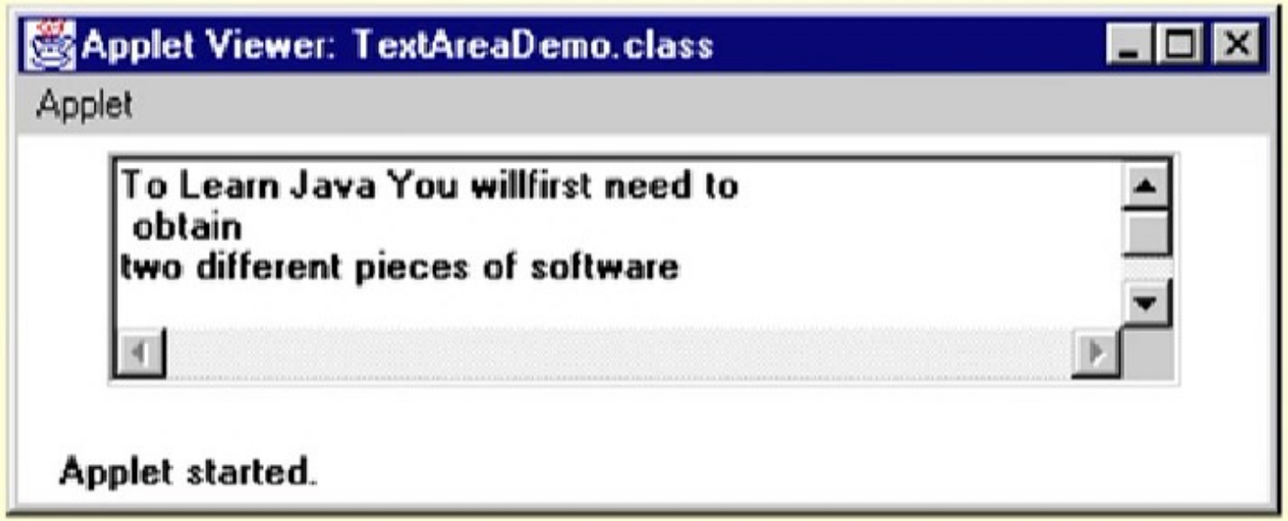
```
import java.awt.*;

class TextFieldExample{
      public static void main(String args[]){
           Frame f = new Frame("TextField Example");
           TextField t1,t2;
           t1 = new TextField("Welcome to IIT Kharagpur.");
           t1.setBounds(50,100, 200,30);
           t2 = new TextField("NPTEL Java Tutorial");
           t2.setBounds(50,150, 200,30);
           f.add(t1); f.add(t2);
           f.setSize(400,400);
           f.setLayout(null);
           f.setVisible(true);
      }
}
```

Creating a TextArea

A TextArea

```
public class TextArea extends Component implements TextComponent
```

The object of a **TextArea** class is a multi line region that displays text. It allows the editing of multiple line text. It inherits `TextComponent` class.



Class TextArea : Constructors

Constructor	Description
TextArea()	Constructs a new text area with the empty string as text.
TextArea(int rows, int columns)	Constructs a new text area with the specified number of rows and columns and the empty string as text.
TextArea(String text)	Constructs a new text area with the specified text.
TextArea(String text, int rows, int columns)	Constructs a new text area with the specified text, and with the specified number of rows and columns.
TextArea(String text, int rows, int columns, int scrollbars)	Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

Class TextArea : Methods

Modifier and Type	*Method*	*Description*
void	append(String str)Appends the given text to the text area's current text.	void
AccessibleContext	getAccessibleContext()	Returns the AccessibleContext associated with this TextArea.
int	getColumns()	Returns the number of columns in this text area.
Dimension	getMinimumSize()	Determines the minimum size of this text area.
Dimension	getMinimumSize(int rows, int columns)	Determines the minimum size of a text area with the specified number of rows and columns.
Dimension	getPreferredSize()	Determines the preferred size of this text area.

Creating a TextArea : An example

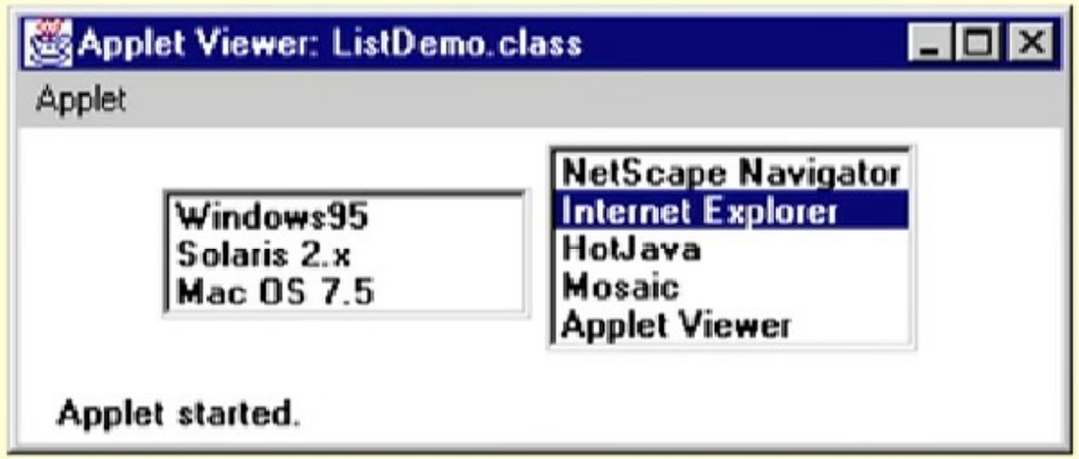
```
import java.awt.*;

public class TextAreaExample{
     TextAreaExample(){
        Frame f = new Frame();
        TextArea area = new TextArea("Welcome to IIT KGP");
        area.setBounds(10,30, 300,300);
        f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }

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

Creating a List

A List

```
public class List extends Component implements ItemSelectable, Accessible
```

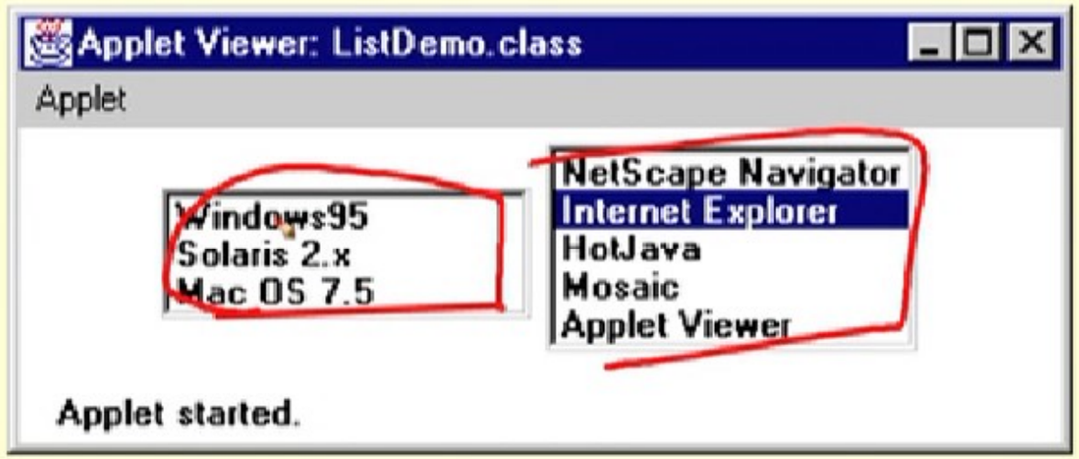
The **List** component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.



A List

```
public class List extends Component implements ItemSelectable, Accessible
```

The **List** component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.

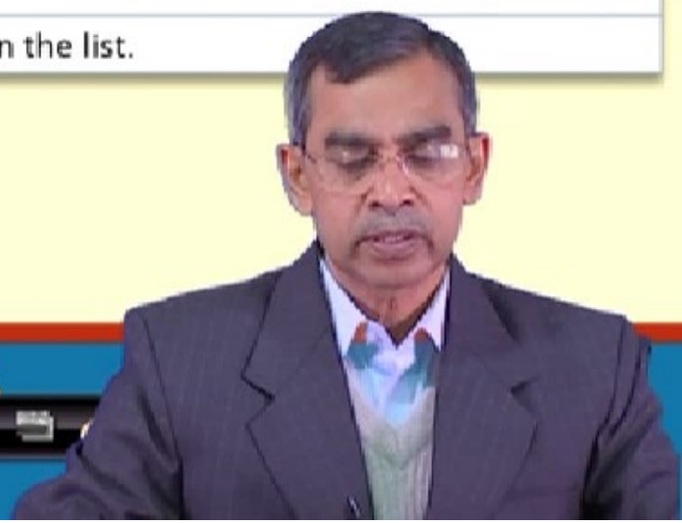


Class List : Constructors

Constructor	Description
List()	Creates a new scrolling list.
List(int rows)	Creates a new scrolling list initialized with the specified number of visible lines.
List(int rows, boolean multipleMode)	Creates a new scrolling list initialized to display the specified number of rows.

Class List : Methods

Modifier and Type	*Method*	*Description*
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this List.
ActionListener[]	getActionListeners()	Returns an array of all the action listeners registered on this list.
String	getItem(int index)	Gets the item associated with the specified index.
int	getItemCount()	Gets the number of items in the list.
ItemListener[]	getItemListeners()	Returns an array of all the item listeners registered on this list.
String[]	getItems()	Gets the items in the list.

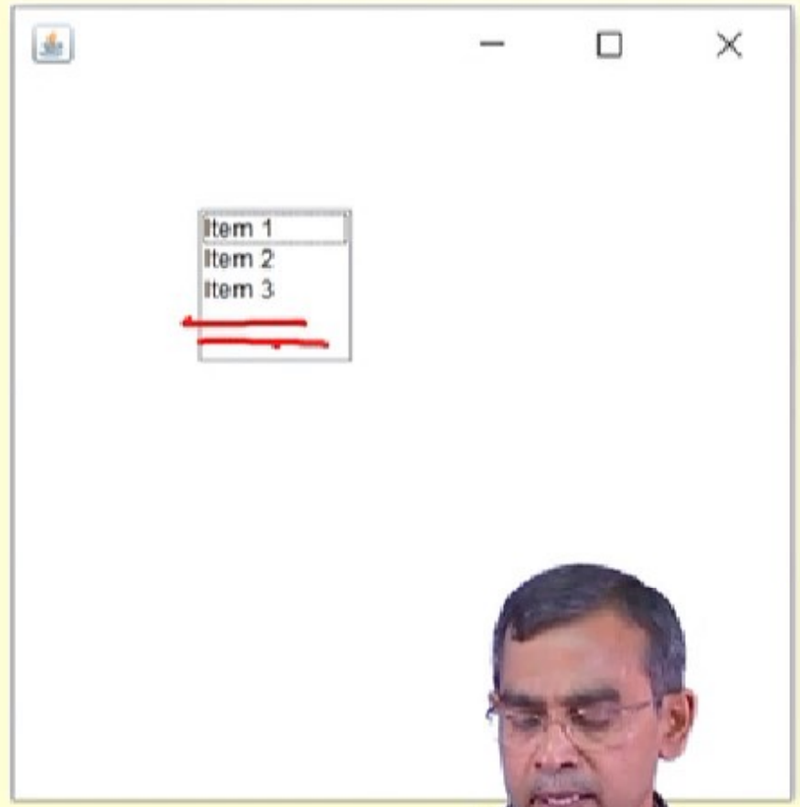


Creating a List : An example

```
import java.awt.*;

public class ListExample{
     ListExample(){
        Frame f = new Frame();
        List l = new List(5);
        l.setBounds(100,100, 75,75);
        l.add("Item 1");
        l.add("Item 2");
        l.add("Item 3");
        f.add(l);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
     public static void main(String args[]){
         new ListExample();
    }
}
```

Creating a List : An example

```
import java.awt.*;

public class ListExample{
     ListExample(){
         Frame f = new Frame();
         List l = new List(5);
         l.setBounds(100,100, 75,75);
         l.add("Item 1");
         l.add("Item 2");
         l.add("Item 3");
         f.add(l);
         f.setSize(400,400);
         f.setLayout(null);
         f.setVisible(true);
     }
     public static void main(String args[]){
          new ListExample();
    }
}
```

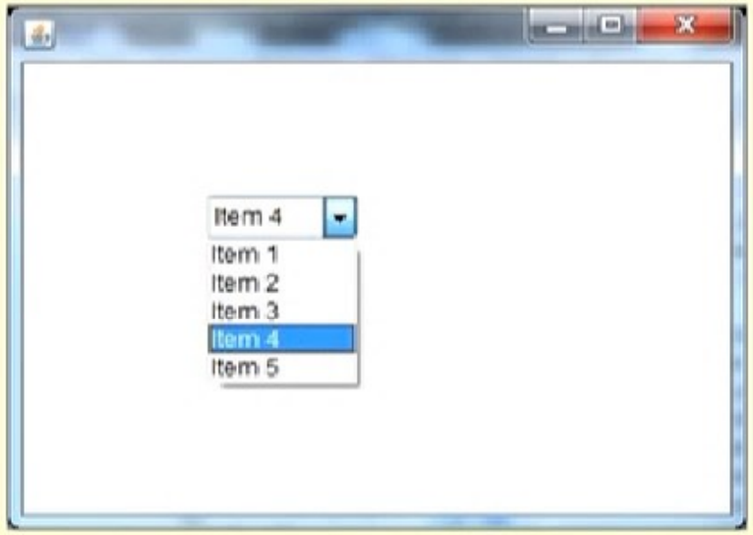


Creating a Choice

A Choice

```
public class Choice extends Component implements ItemSelectable, Accessible
```

The object of **Choice** class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.



Class Choice : Constructors

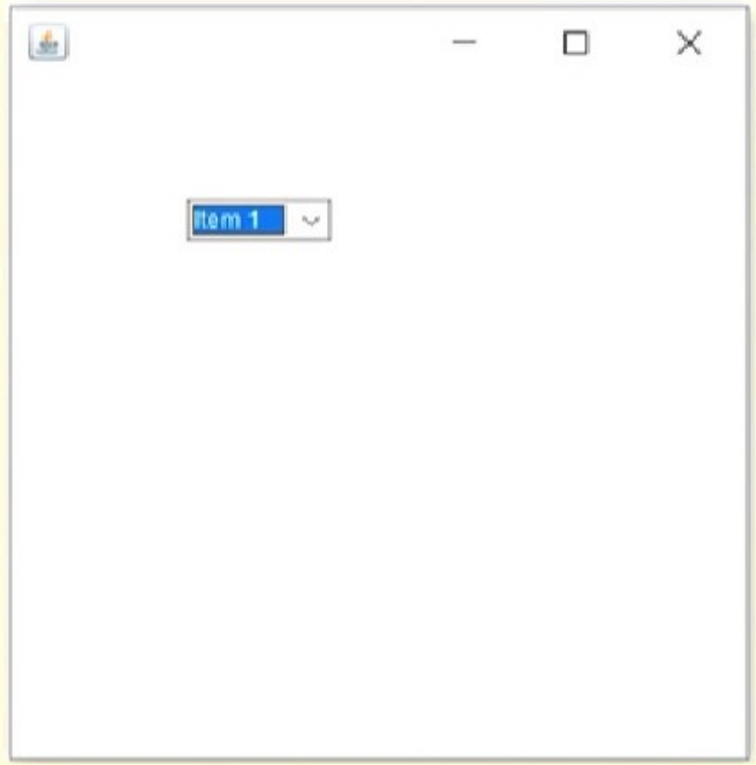
Constructor	Description
Choice ( )	Creates a new choice menu.

Class Choice : Methods

Modifier and Type	*Method*	*Description*
void	addItemListener(ItemListener l)	Adds the specified item listener to receive item events from this check box.
void	addNotify()	Creates the peer of the Checkbox.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Checkbox.
CheckboxGroup	getCheckboxGroup()	Determines this check box's group.
ItemListener[]	getItemListeners()	Returns an array of all the item listeners registered on this checkbox.
String	getLabel()	Gets the label of this check box.
<T extends EventListener> T[]	getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as *Foo*Listeners upon this Checkbox.
Object[]	getSelectedObjects()	Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
boolean	getState()	Determines whether this check box is in the "on" or "off" state.
protected String	paramString()	Returns a string representing the state of this Checkbox.
protected void	processEvent(AWTEvent e)	Processes events on this check box.
protected void	processItemEvent(ItemEvent e)	Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
void	removeItemListener(ItemListener l)	Removes the specified item listener so that the item listener no longer receives item events from this check box.
void	setCheckboxGroup(CheckboxGroup g)	Sets this check box's group to the specified check box group.
void	setLabel(String label)	Sets this check box's label to be the string argument.
void	setState(boolean state)	Sets the state of this check box to the specified state.

Creating a Choice : An example

```
import java.awt.*;
public class ChoiceExample{
        ChoiceExample(){
        Frame f= new Frame();
        Choice c=new Choice();
        c.setBounds(100,100, 75,75);
        c.add("Item 1");
        c.add("Item 2");
        c.add("Item 3");
        c.add("Item 4");
        c.add("Item 5");
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[]){
        new ChoiceExample();
    }
}
```

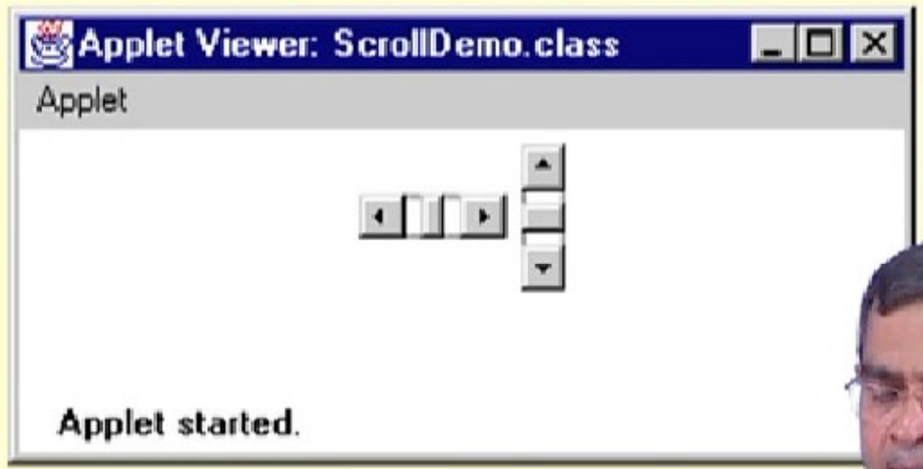


Creating a Scrollbar

A Scrollbar

```
public class Scrollbar extends Component implements Adjustable, Accessible
```

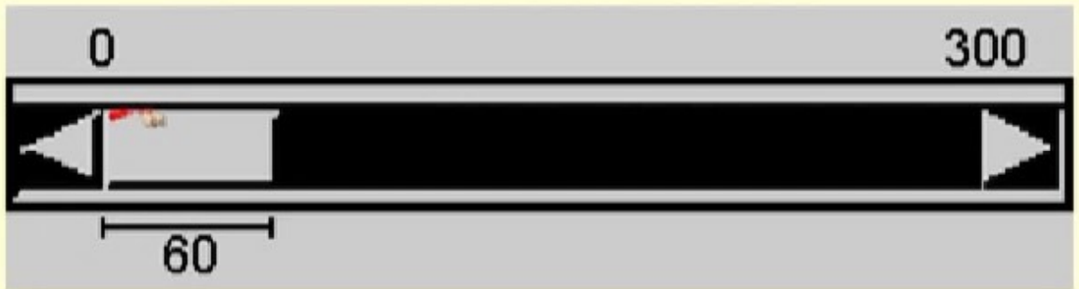
The **Scrollbar** class embodies a scroll bar, a familiar user-interface object. A scrollbar provides a convenient means for allowing a user to select from a range of values.

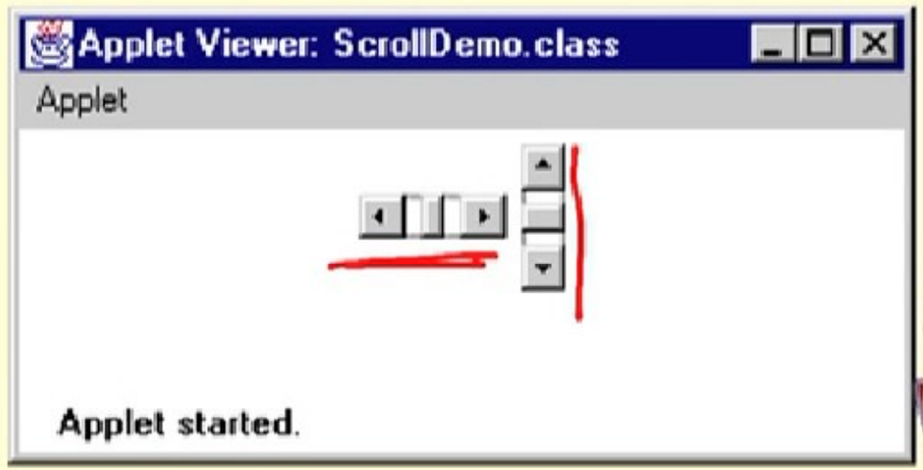


A Scrollbar

```
public class Scrollbar extends Component implements Adjustable, Accessible
```

The **Scrollbar** class embodies a scroll bar, a familiar user-interface object. A scrollbar provides a convenient means for allowing a user to select from a range of values.





Class Scrollbar : Constructors

Constructor	Description
Scrollbar()	Constructs a new vertical scroll bar.
Scrollbar(int orientation)	Constructs a new scroll bar with the specified orientation.
Scrollbar(int orientation, int value, int visible, int minimum, int maximum)	Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Class Scrollbar : Methods

Modifier and Type	*Method*	*Description*
void	addAdjustmentListener(AdjustmentListener l)	Adds the specified adjustment listener to receive instances of AdjustmentEvent from this scroll bar.
void	addNotify()	Creates the Scrollbar's peer.
AccessibleContext	getAccessibleContext()	Gets the AccessibleContext associated with this Scrollbar.
AdjustmentListener[]	getAdjustmentListeners()	Returns an array of all the adjustment listeners registered on this scrollbar.
int	getBlockIncrement()	Gets the block increment of this scroll bar.
int	getMaximum()	Gets the maximum value of this scr

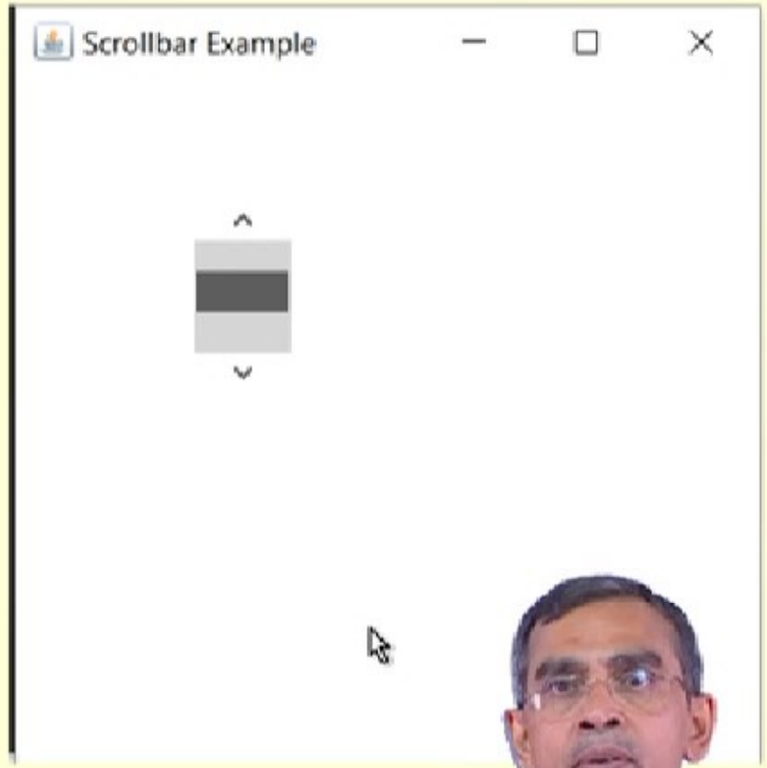
Creating a Scrollbar : An example

Class Scrollbar : Methods

Modifier and Type	*Method*	*Description*
int	getMinimum()	Gets the minimum value of this scroll bar.
int	getOrientation()	Returns the orientation of this scroll bar.
void	setValue(int newValue)	Sets the value of this scroll bar to the specified value.
int	getUnitIncrement()	Gets the unit increment for this scrollbar.
int	getValue()	Gets the current value of this scroll bar.
boolean	getValueIsAdjusting()	Returns true if the value is in the process of changing as a result of actions being taken by the user.

Creating a Scrollbar : An example

```
import java.awt.*;

class ScrollbarExample{
     ScrollbarExample(){
          Frame f= new Frame("Scrollbar Example");
          Scrollbar s = new Scrollbar();
          s.setBounds(100,100, 50,100);
          f.add(s);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
     }
     public static void main(String args[]){
          new ScrollbarExample();
     }
}
```



Thank You

Lecture 40
A W T Programming - II

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

Java AWT Programming – II

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

GUIs with Layout Managers

Layout Managers

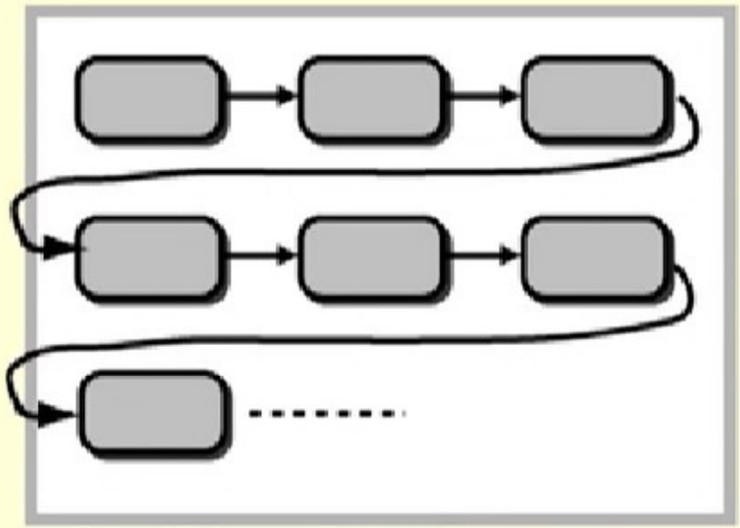
- It is a tedious process (particularly with a large number of components) to place components on the screen (it has to be specify explicitly).
- To tackle this, AWT includes the notion of layout managers: the layout of components in a container may be governed by a layout manager.
- A `Container` in Java is nothing but an applet which generally contains number of GUI attributes such as `Frame`, `Panel`, `Label`, `Button`, etc.
- Each `Container` object has a layout manger which is an instance of any class that implements the `Layout Manager` interface.
- Each component, such as a `Panel`, or a `Frame` has a default layout manager associated with it which may be changed by the Java developer when an instance of that container is created.
- Each layout manager keeps track of a list of `Components`. The layout manager is notified each time one adds a `Component` to a `Panel`, for example.
- Java has the following five predefined layout manager classes :

FlowLayout ***BorderLayout*** ***GridLayout*** ***CardLayout***

FlowLayout Manager

FlowLayout Manager

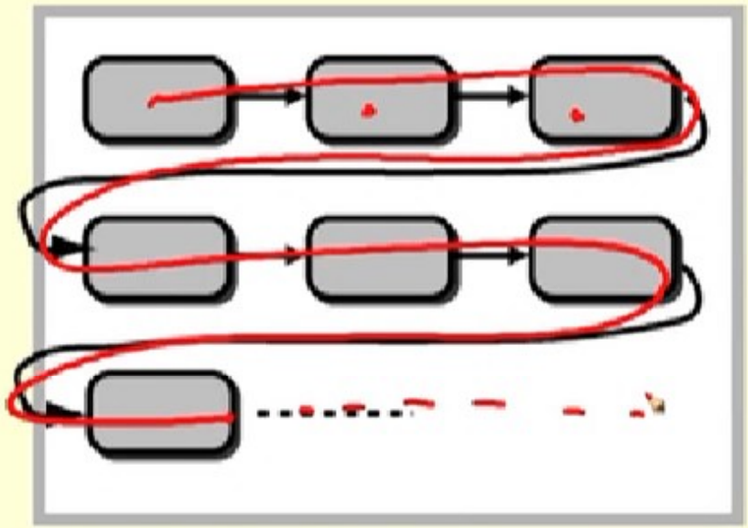
The `FlowLayout` is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.



Components Added by FlowLayout

FlowLayout Manager

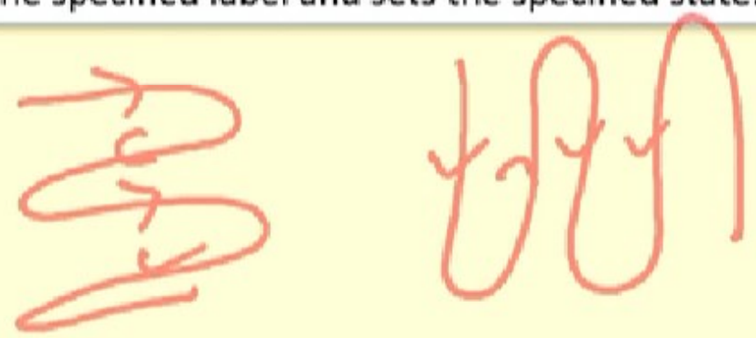
The `FlowLayout` is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.



Components Added by FlowLayout

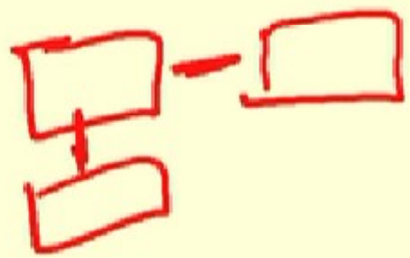
Class FlowLayout : Constructor

Constructor	Description
FlowLayout()	Creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align)	Creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align, int hgap, int vgap)	Creates a check box with the specified label and sets the specified state.



Class FlowLayout : Constructor

Constructor	Description
FlowLayout()	Creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align)	Creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align, int hgap, int vgap)	Creates a check box with the specified label and sets the specified state.

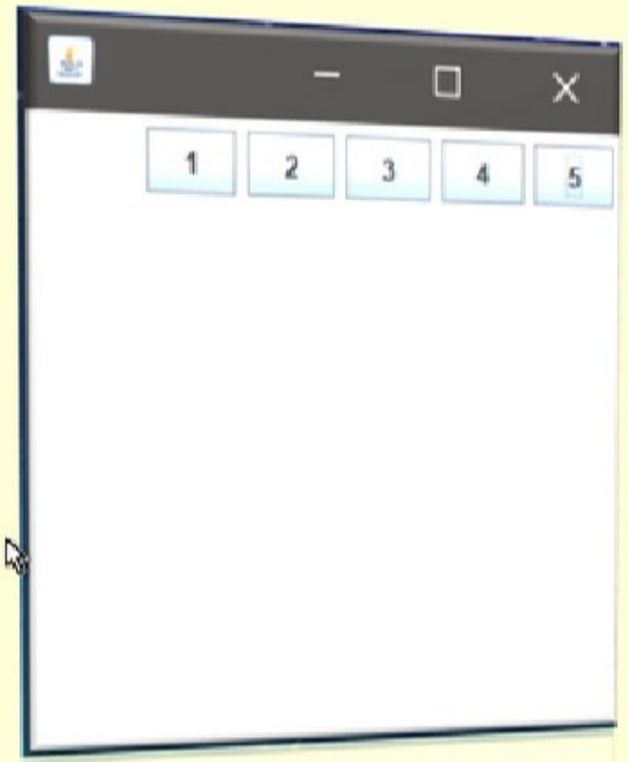


Class FlowLayout : Fields

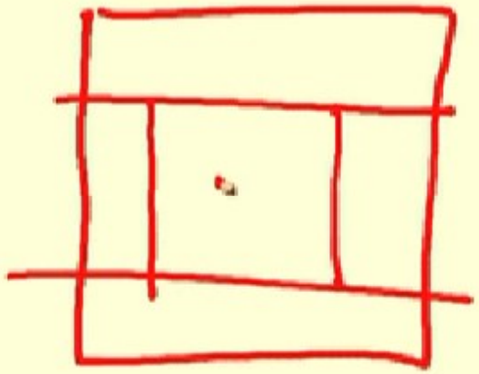
Modifier and Type	*Method*	*Description*
static int	CENTER	This value indicates that each row of components should be centered.
static int	LEADING	This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
static int	LEFT	This value indicates that each row of components should be left-justified.
static int	RIGHT	This value indicates that each row of components should be right-justified.
static int	TRAILING	This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

Creating a FlowLayout : An example

```
import java.awt.*;

public class MyFlowLayout{
Frame f;
    MyFlowLayout(){
            f = new Frame();
            Button b1 = new Button("1");
            Button b2 = new Button("2");
            Button b3 = new Button("3");
            Button b4 = new Button("4");
            Button b5 = new Button("5");
            f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
            f.setLayout(new FlowLayout(FlowLayout.RIGHT));
            //setting flow layout of right alignment
            f.setSize(300,300);
            f.setVisible(true);
    }
    public static void main(String[] args) {
         new MyFlowLayout();
    }
}
```

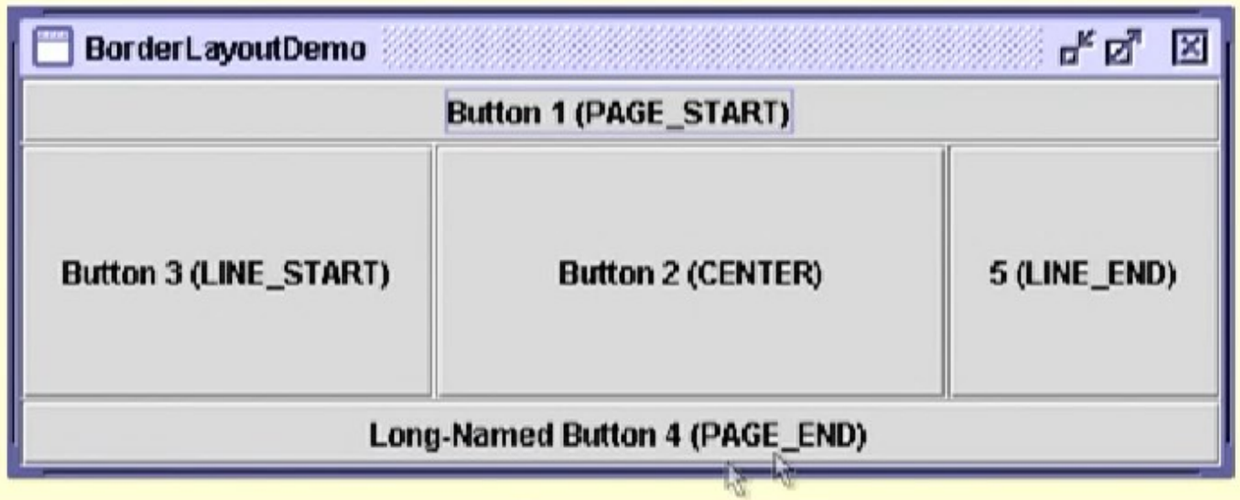


BorderLayout Manager



BorderLayout Manager

The `BorderLayput` manager is used to arrange components in a particular manner. This layout manager arranges components to be placed around borders: East, West, North, South and Center.



Class BorderLayout : Constructor

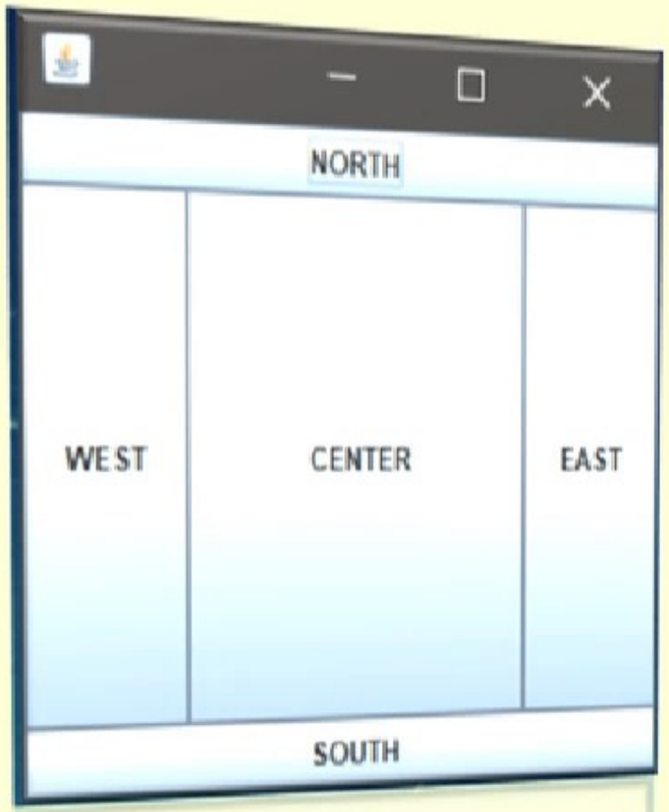
Constructor	Description
`BorderLayout()`	Creates a border layout but with no gaps between the components.
`JBorderLayout(int hgap, int vgap)`	Creates a border layout with the given horizontal and vertical gaps between the components.
`FlowLayout(int align, int hgap, int vgap)`	Creates a check box with the specified label and sets the specified state.

Class BorderLayout : Fields

Modifier and Type	*Method*	*Description*
static String	CENTER	The center layout constraint (middle of container).
static String	EAST	The east layout constraint (right side of container).
static String	NORTH	The north layout constraint (top of container).
static String	SOUTH	The south layout constraint (bottom of container).
static String	WEST	The west layout constraint (left side of container).

Creating a BorderLayout : An example

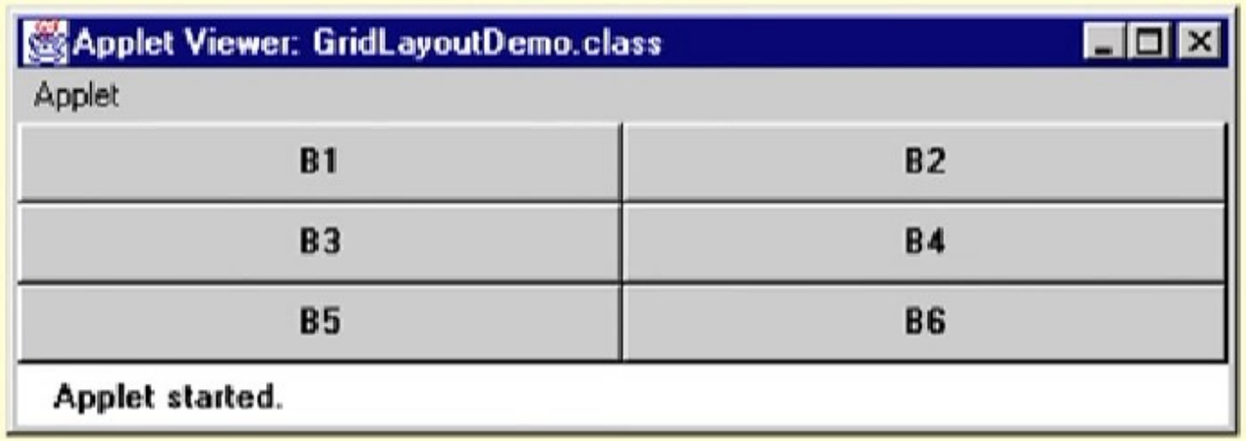
```
import java.awt.*;
  public class MyBorderLayout {
    Frame f;
    MyBorderLayout(){
          f = new Frame();
          Button b1 = new Button("NORTH");;
          Button b2 = new Button("SOUTH");;
          Button b3 = new Button("EAST");;
          Button b4 = new Button("WEST");;
          Button b5 = new Button("CENTER");;
          f.add(b1,BorderLayout.NORTH);
          f.add(b2,BorderLayout.SOUTH);
          f.add(b3,BorderLayout.EAST);
          f.add(b4,BorderLayout.WEST);
          f.add(b5,BorderLayout.CENTER);
          f.setSize(300,300);   f.setVisible(true);
      }
    public static void main(String[] args) {
        new MyBorderLayout();
    }

}
```



GridLayout Manager

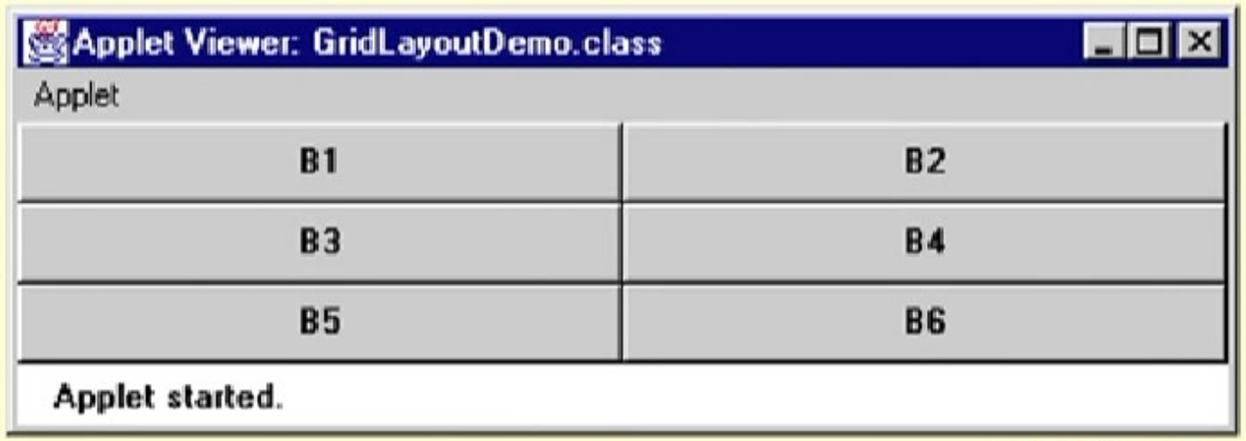
GridLayout Manager

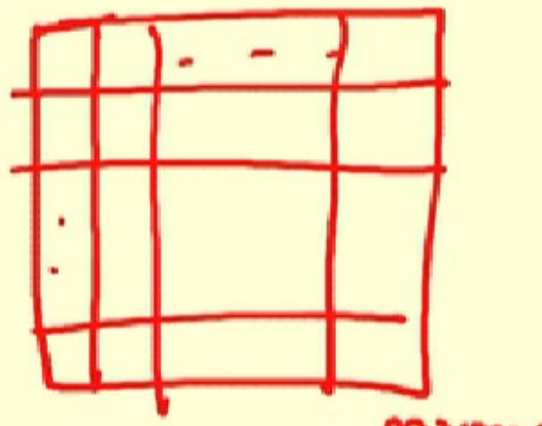
The **GridLayout** manager is used to arrange the components in rectangular grid. One component is displayed in each rectangle.



GridLayout Manager

The **GridLayout** manager is used to arrange the components in rectangular grid. One component is displayed in each rectangle.





Class GridLayout : Constructor

Constructor	Description
GridLayout()	Creates a grid layout with one column per component in a row.
GridLayout(int rows, int columns)	Creates a grid layout with the given rows and columns but no gaps between the components.
GridLayout(int rows, int columns, int hgap, int vgap)	Creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Creating a GridLayout : An example

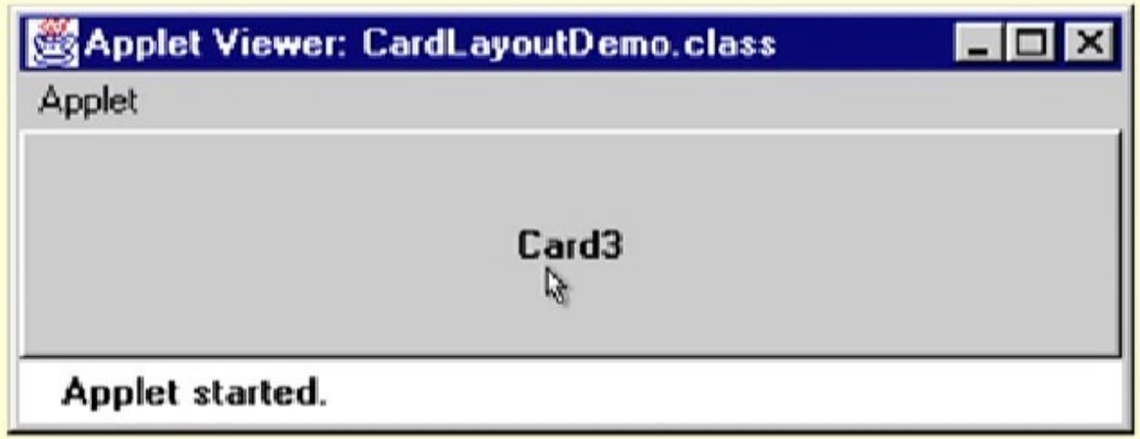
```
import java.awt.*;
public class GridLayoutDemo{
    Frame f;
    GridLayoutDemo(){
            f = new Frame();
            Button b1 = new Button("1");
            Button b2 = new Button("2");
            Button b3 = new Button("3");
            Button b4 = new Button("4");
            Button b5 = new Button("5");
            Button b6 = new Button("6");
            Button b7 = new Button("7");
            Button b8 = new Button("8");
            Button b9 = new Button("9");
            f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
            f.add(b6);f.add(b7);f.add(b8);f.add(b9);
            f.setLayout(new GridLayout(3,3));//3 rows and 3 columns
            f.setSize(300,300);
            f.setVisible(true);}
    public static void main(String[] args) {
         new GridLayoutDemo();
    }
}
```

CardLayout Manager

CardLayout Manager

The `CardLayout` manager manages the components in such a manner that only one component is visible at a time.

It treats each component as a card that is why it is known as **CardLayout**.



Class CardLayout : Constructor

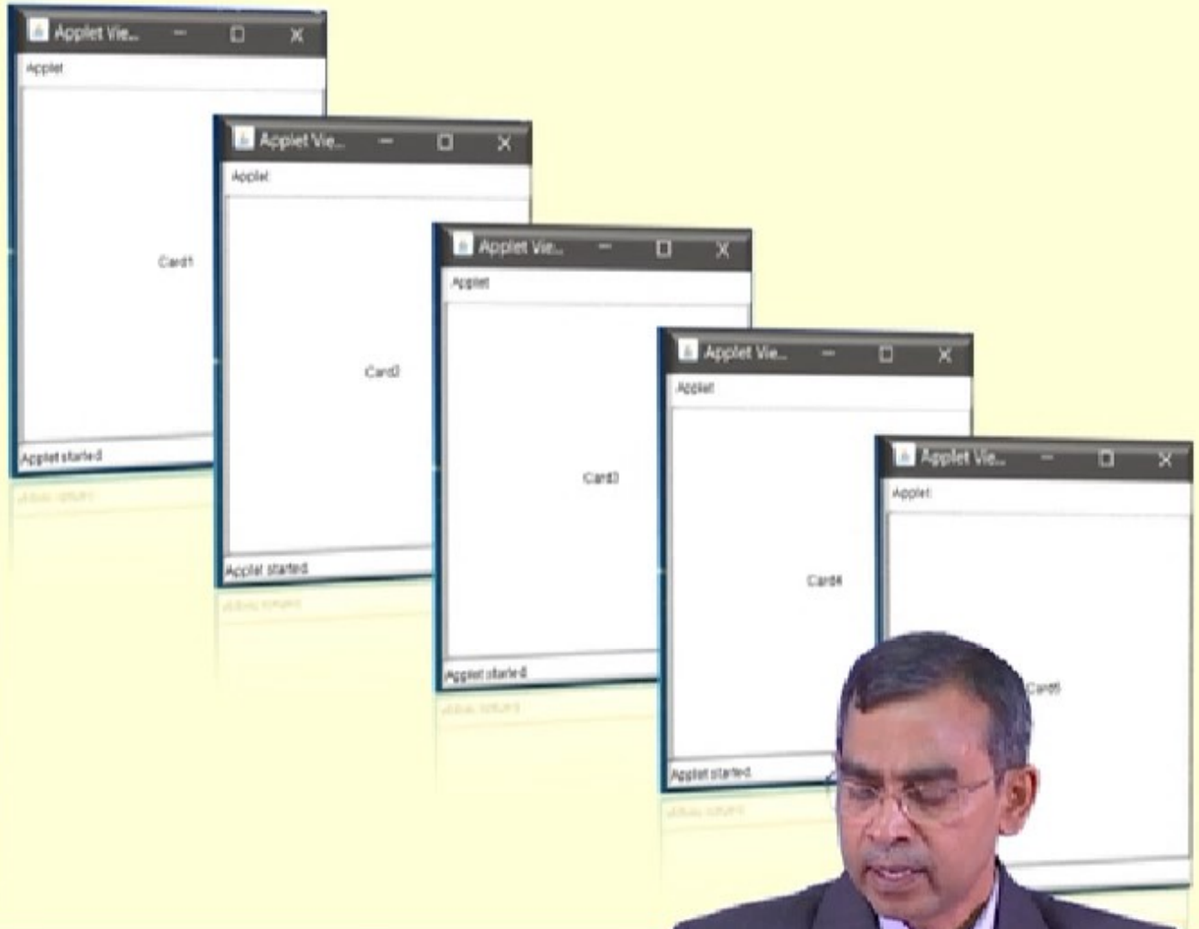
Constructor	Description
`CardLayout()`	Creates a card layout with zero horizontal and vertical gap.
`CardLayout(int hgap, int vgap)`	Creates a card layout with the given horizontal and vertical gap.

Class CardLayout : Methods

Modifier and Type	*Method*	*Description*
public void	next(Container parent)	Is used to flip to the next card of the given container.
public void	previous(Container parent)	Is used to flip to the previous card of the given container.
public void	first(Container parent)	Is used to flip to the first card of the given container.
public void	last(Container parent)	Is used to flip to the last card of the given container.
public void	show(Container parent, String name)	Is used to flip to the specified card with the given name.

Creating a CardLayout Manager

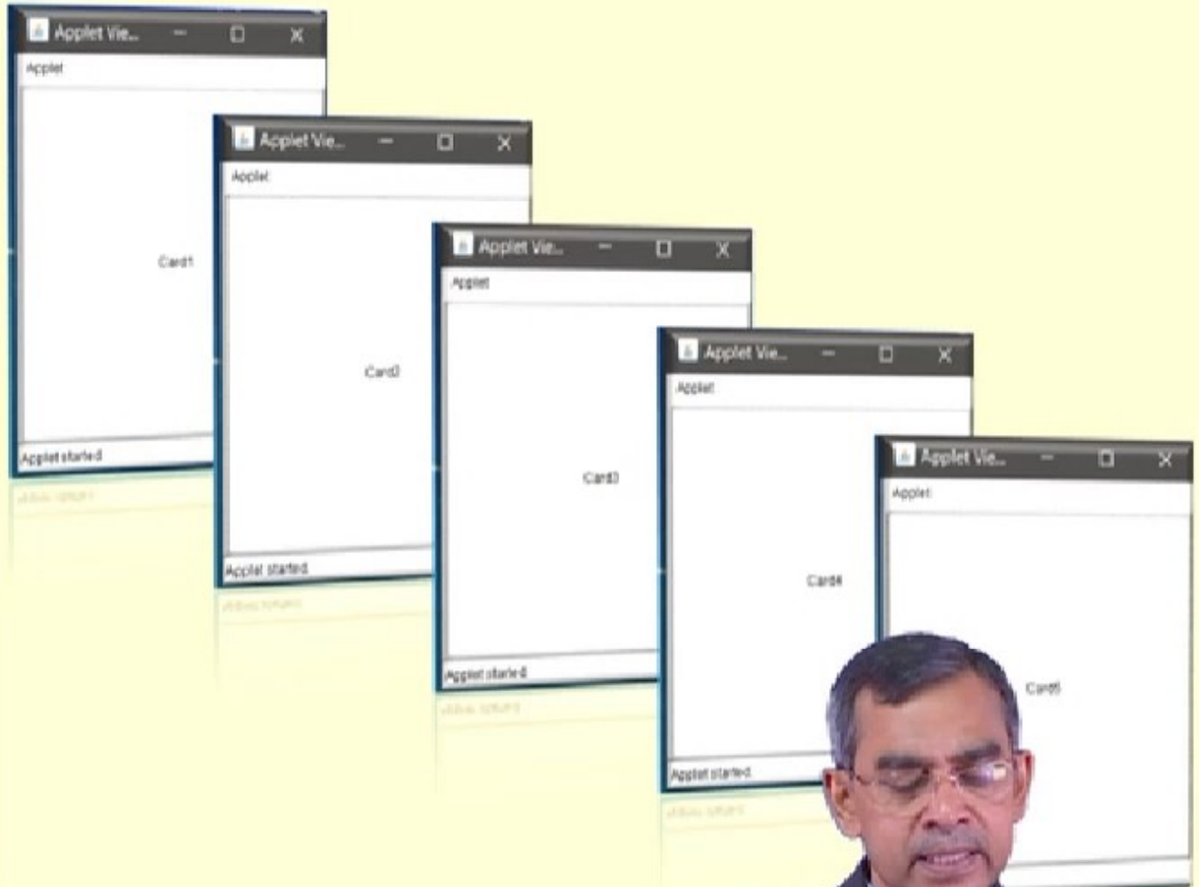
```
import java.awt.*;

public class Cards extends java.applet.Applet {
    CardLayout   layout;
    public void init ( ) {
            layout = new CardLayout ( ) ;
            setLayout (layout );
            add ("1", new Button ("Card1" ));
            add ("2", new Button ("Card2" ));
            add ("3", new Button ("Card3" ));
            add ("4", new Button ("Card4" ));
            add ("5", new Button ("Card5" ));
    }
    public boolean keyDown (Event e, int key ){
            layout.next(this);
            return (true );
    }
}
```



5 Sec. Backward - [00:18:28 / 52%]

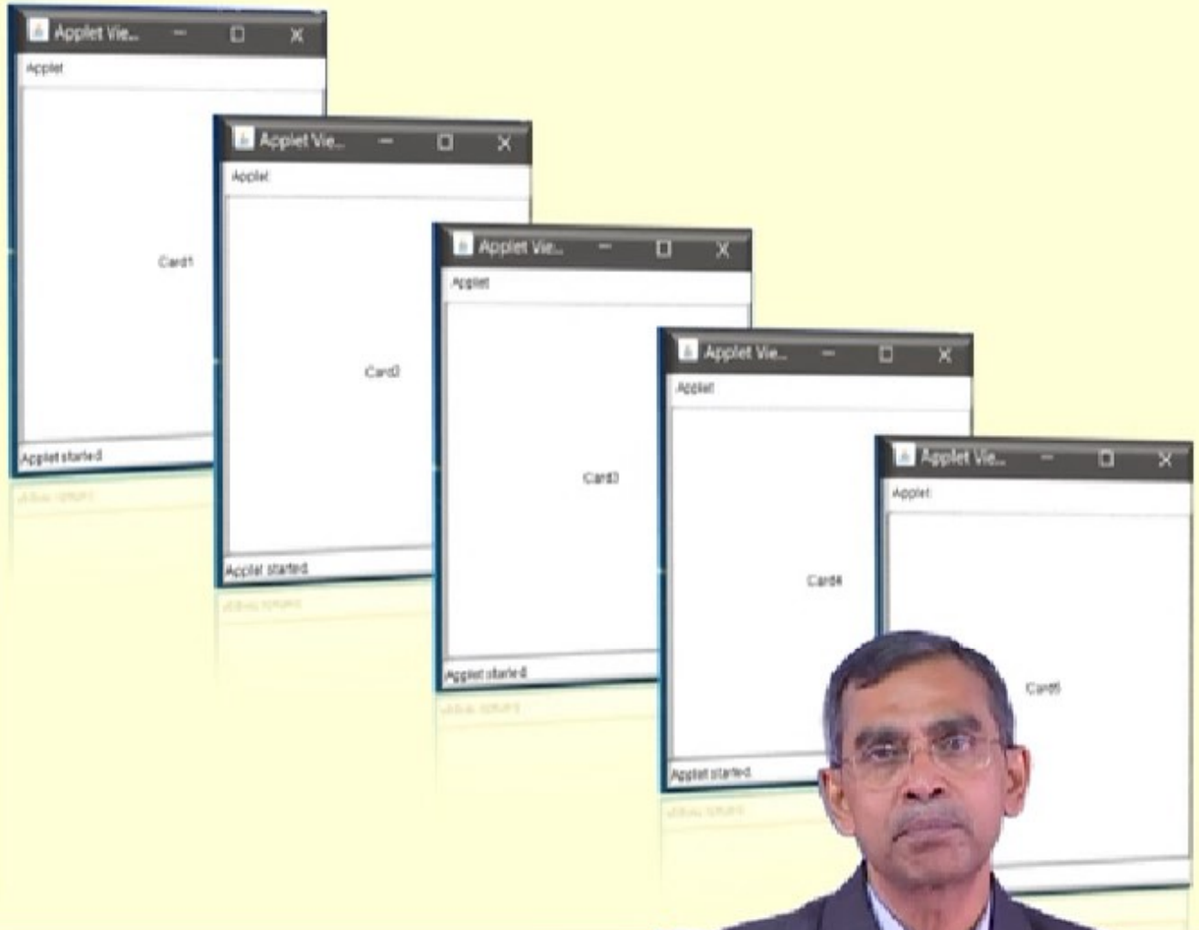
Creating a CardLayout Manager

```
import java.awt.*;

public class Cards extends java.applet.Applet {
    CardLayout   layout;
    public void init ( ) {
            layout = new CardLayout ( ) ;
            setLayout (layout );
            add ("1", new Button ("Card1" ));
            add ("2", new Button ("Card2" ));
            add ("3", new Button ("Card3" ));
            add ("4", new Button ("Card4" ));
            add ("5", new Button ("Card5" ));
    }
    public boolean keyDown (Event e, int key ){
            layout.next(this);
            return (true );
    }
}
```



Creating a CardLayout Manager

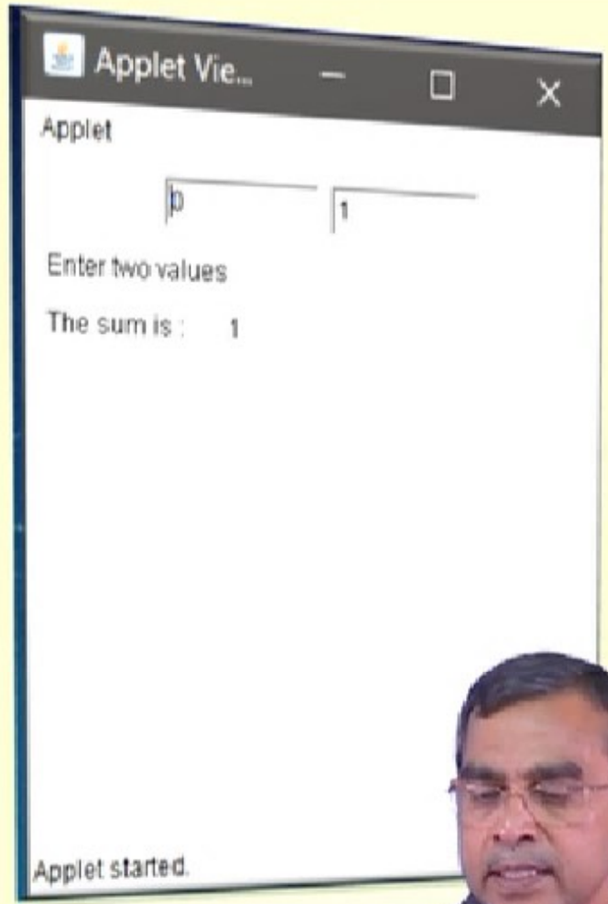
```
import java.awt.*;

public class Cards extends java.applet.Applet {
    CardLayout   layout;
    public void init ( ) {
            layout = new CardLayout ( ) ;
            setLayout (layout );
            add ("1", new Button ("Card1" ));
            add ("2", new Button ("Card2" ));
            add ("3", new Button ("Card3" ));
            add ("4", new Button ("Card4" ));
            add ("5", new Button ("Card5" ));
    }
    public boolean keyDown (Event e, int key ){
            layout.next(this);
            return (true );
    }
}
```



Application : Interactive Applet

Interactive applet

```
import java.applet.*;
import java.awt.*;
public class InteractiveApplet extends Applet {
    TextField inputA, inputB;
    public void init ( ) {
                inputA = new TextField(8);
                inputB = new TextField(8);
                add(inputA); add(inputB);
                inputA.setText("0");    inputB.setText("1");
    }
    public void paint(Graphics g) {
        int x = 0; int y = 0; int z = 0;
        String s;
        g.drawString("Enter two values", 10,50);
        try {
           s = inputA.getText(); x = Integer.parseInt(s);
           s = inputB.getText(); y = Integer.parseInt(s);
           z = x+y; s = String.valueOf(z); g.drawString("The sum is :",10,75);
           g.drawString(s, 100,75);
                           }catch(Exception e) { }
            }
}
```



Application : Event Handling

Event Handling : The concept

- To add life to the components
-
- Java GUI supports event-driven programming

- The device usually handled are:
  - Mouse
  - Keyboard

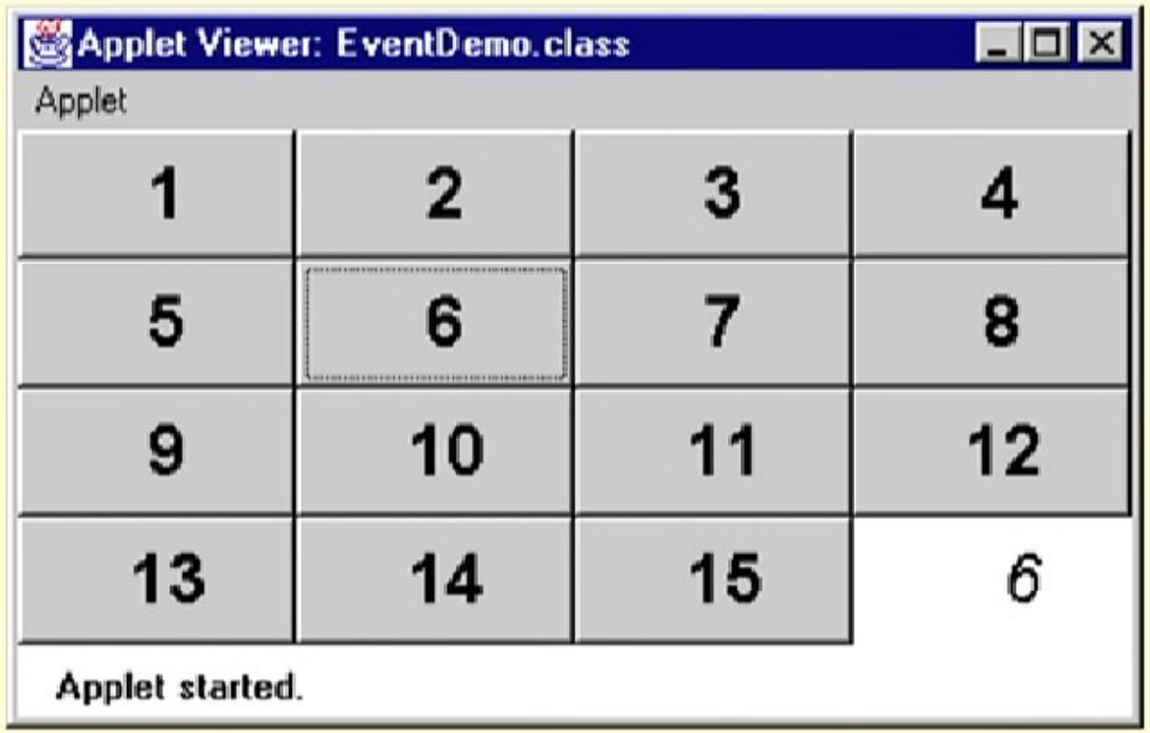
Event Handling : The concept

- To add life to the components
-
- Java GUI supports event-driven programming

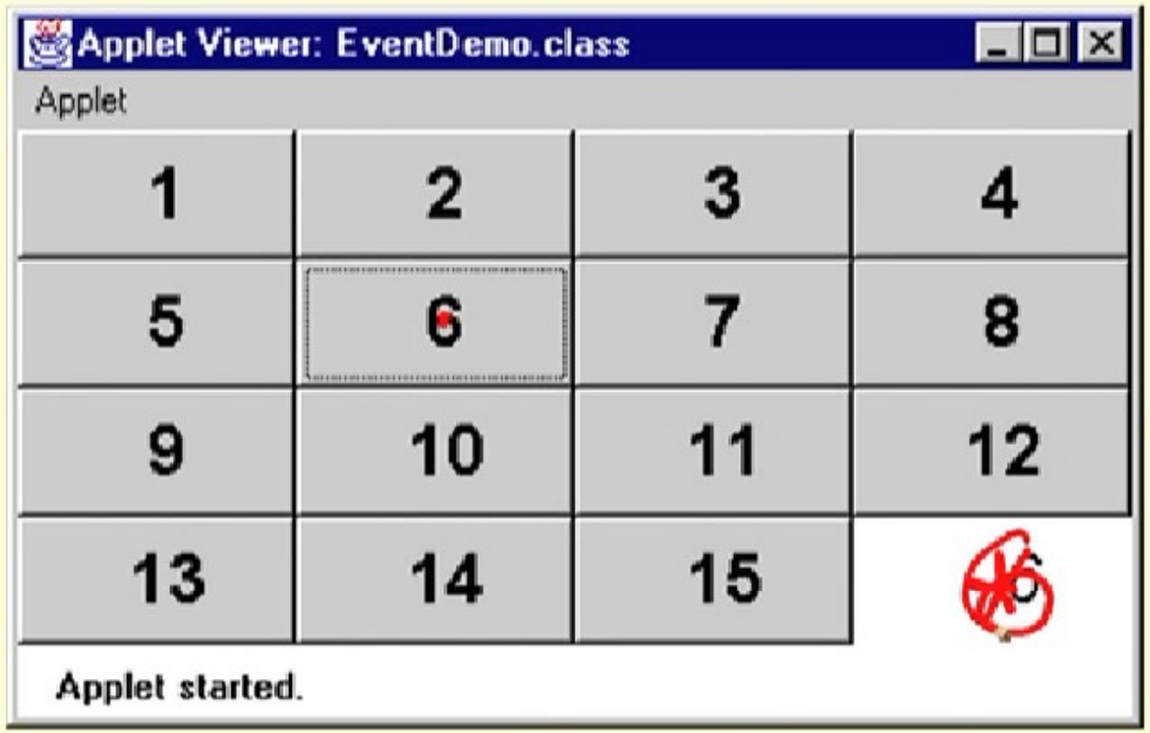
- The device usually handled are:
  - Mouse
  - Keyboard

- Possible events with these devices are
  - Mouse events
    - Moving, gragging, entering, exiting
  - Keyboard events
    - Pressing and releasing
  - Windows events
    - Destroying, exposing, iconifying, deiconifying and moving
  - Scrolling events
    - Scrolling line, page, location (to a point)

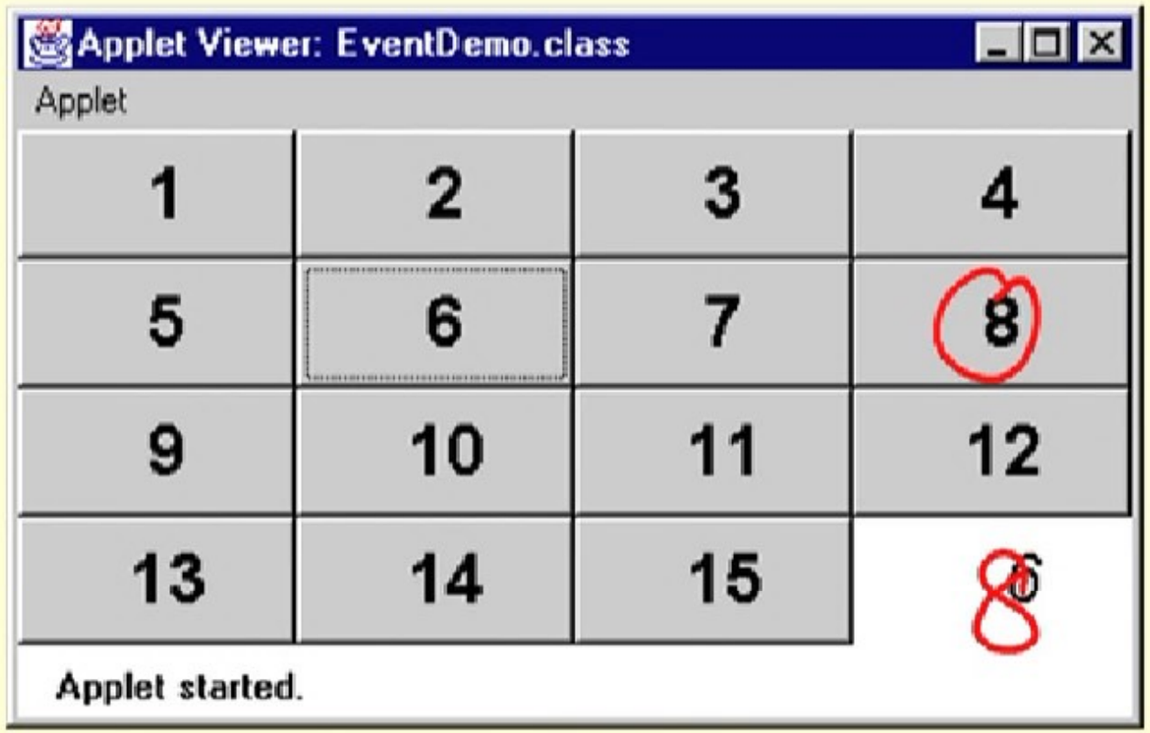
Event handling : An example



Event handling : An example



Event handling : An example



Event handling : An example

```
public class EventDemo extends Applet {
    static final int n = 4;
    Label label;
    public void init(  ) {
            setLayout (new GridLayout (n,n));
            setFont (new Font ("Helvetica",  Font.BOLD, 24 ));
            for (int i = 0; i < n; i++ )        {
               for (int j = 0; j < n; j++ ) {
                  int k = i * n + j+1;
                  if (k < 16)
                     add (new Button ("  " + k+1);
               }
            }
            label = new Label (" * " , Label.CENTER );
            label.setFont (new FONT (" Times Roman", Font.ITALIC, 24 ));
            add (label );
    }
```

Event handling : An example

```
public class EventDemo extends Applet {
    static final int n = 4;
    Label label;
    public void init(  ) {
            setLayout (new GridLayout (n,n));
            setFont (new Font ("Helvetica",  Font.BOLD, 24 ));
            for (int i = 0; i < n; i++ )          {
               for (int j = 0; j < n; j++ ) {
                  int k = i * n + j+1;
                  if (k < 16)
                     add (new Button ("  " + k+1);
               }
            }
            label = new Label (" * " , Label.CENTER );
            label.setFont (new FONT (" Times Roman", Font.ITALIC, 24 ));
            add (label );
    }
```

Event handling : An example

```
public class EventDemo extends Applet {
    static final int n = 4;
    Label label;
    public void init(  ) {
            setLayout (new GridLayout (n,n));
            setFont (new Font ("Helvetica",  Font.BOLD, 24 ));
            for (int i = 0; i < n; i++ )          {
               for (int j = 0; j < n; j++ ) {
                  int k = i * n + j+1;
                  if (k < 16)
                     add (new Button ("   " + k+1);
               }
            }
            label = new Label (" * " , Label.CENTER );
            label.setFont (new FONT (" Times Roman", Font.ITAL
            add (label );
    }
```

```
    //  Overridden event handler
    public boolean action(Event e, Object obj) {

            //  Wait for clicking a button object
            if (e.target instanceof  Button ) ) {

            //  Print the value of the Clicked button
               label.setText ((String ) obj );

            }
            return false ;
    }
}
```

Event handling : An example

```
public class EventDemo extends Applet {
    static final int n = 4;
    Label label;
    public void init(  ) {
            setLayout (new GridLayout (n,n));
            setFont (new Font ("Helvetica",  Font.BOLD, 24 ));
            for (int i = 0; i < n; i++ )          {
               for (int j = 0; j < n; j++ ) {
                  int k = i * n + j+1;
                  if (k < 16)
                     add (new Button ("   " + k+1);
               }
            }
            label = new Label (" * " , Label.CENTER );
            label.setFont (new FONT (" Times Roman", Font.ITAL
            add (label );
    }
```

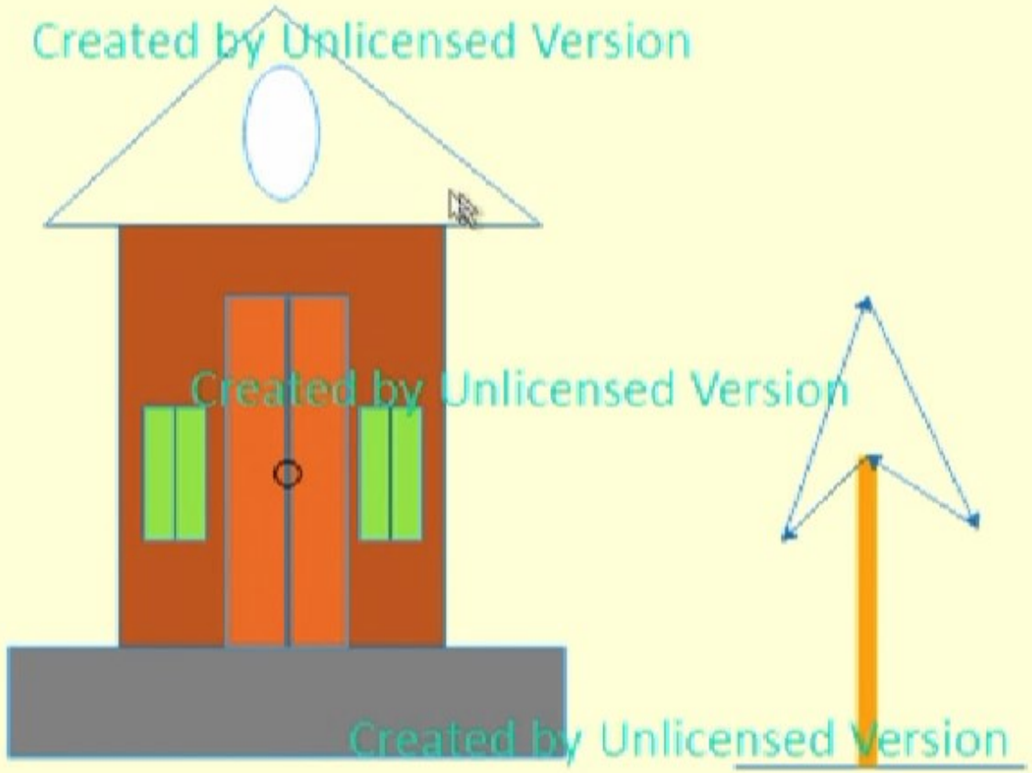
```
    //  Overridden event handler
    public boolean action(Event e, Object obj) {

            //  Wait for clicking a button object
            if (e.target instanceof  Button ) ) {

            //  Print the value of the Clicked button
               label.setText ((String ) obj );

            }
            return false ;
    }
}
```

Application : Graphics

Class Graphics: Some methods

```
public void drawLine (x1, y1, x2, y2 );
public void drawRect (x,  y,  width, height );
public void fillRect (x, y,  width, height );
public void clearRect (x,y, width,  height );
public void drawRoundRect (x, y, width, height  arcWidth, arcHeight );
public void fillRoundRect (x, y, width, height  arcWidth, arcHeight );
public void draw3DRect (x, y, width, height, boolean raised );
public void fill3DRect (x, y, width, height, boolean raised );
public void drawOval (x, y, width, height );
public void  fillOval (x, y, width, height );
public void drawArc (x, y, width, height  startAngle, arcangle );
public void fillArc (x, y,  width, height  startAngle, arcAngle );
public void drawPolygon (int [ ], xPoints, int [ ] yPoints );
public void drawString ( String s, x, y);
public void drawChars (char data [ ], offset, length, x, y );
public void drawBytes ( byte data [ ], offset, length, x, y);
public void drawBytes ( byte data [ ], offset, length, x,y );
```

Graphics : An example



Graphics : An example