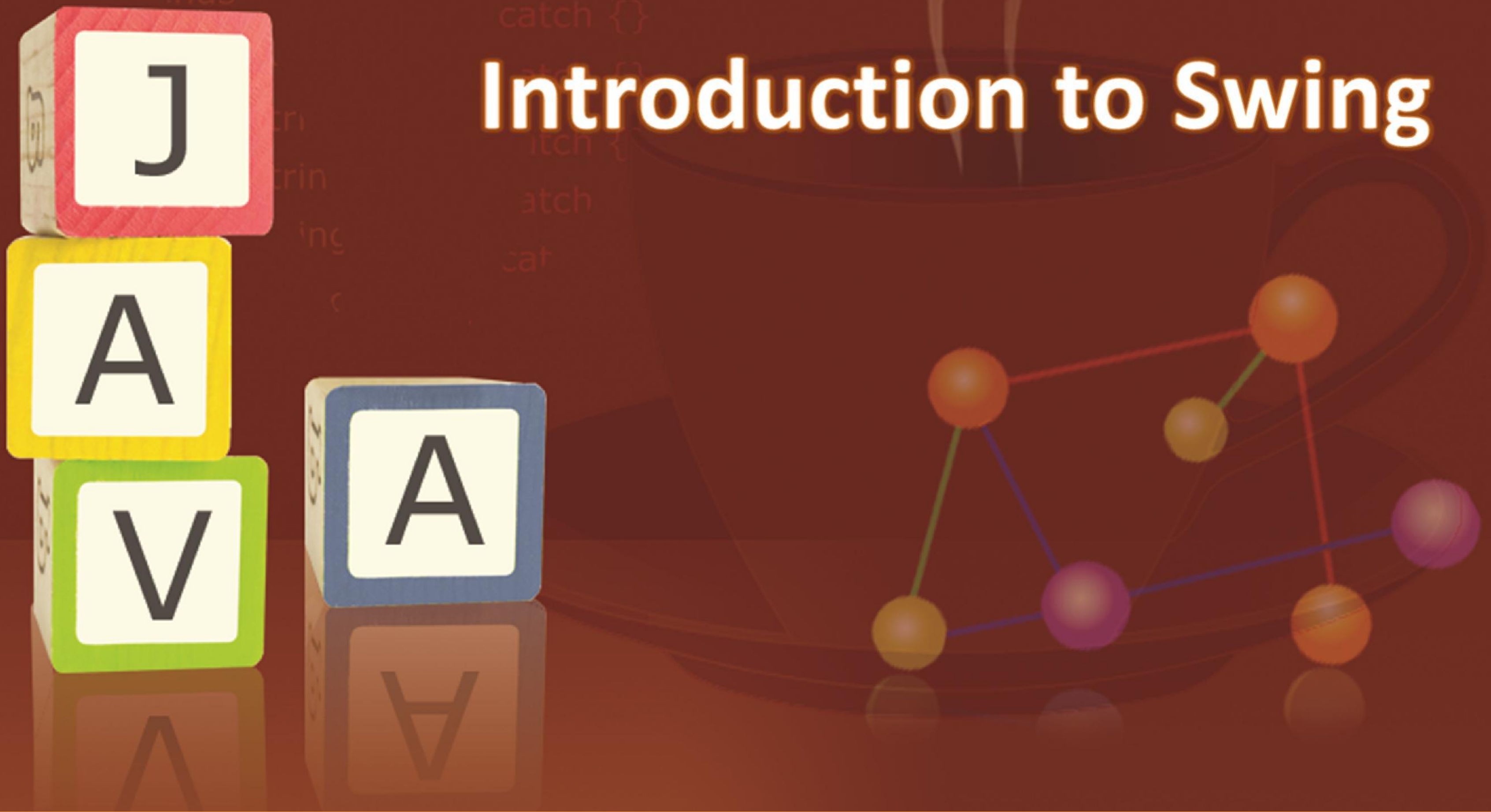
Distributed Programming in Java

Session: 1

Introduction to Swing

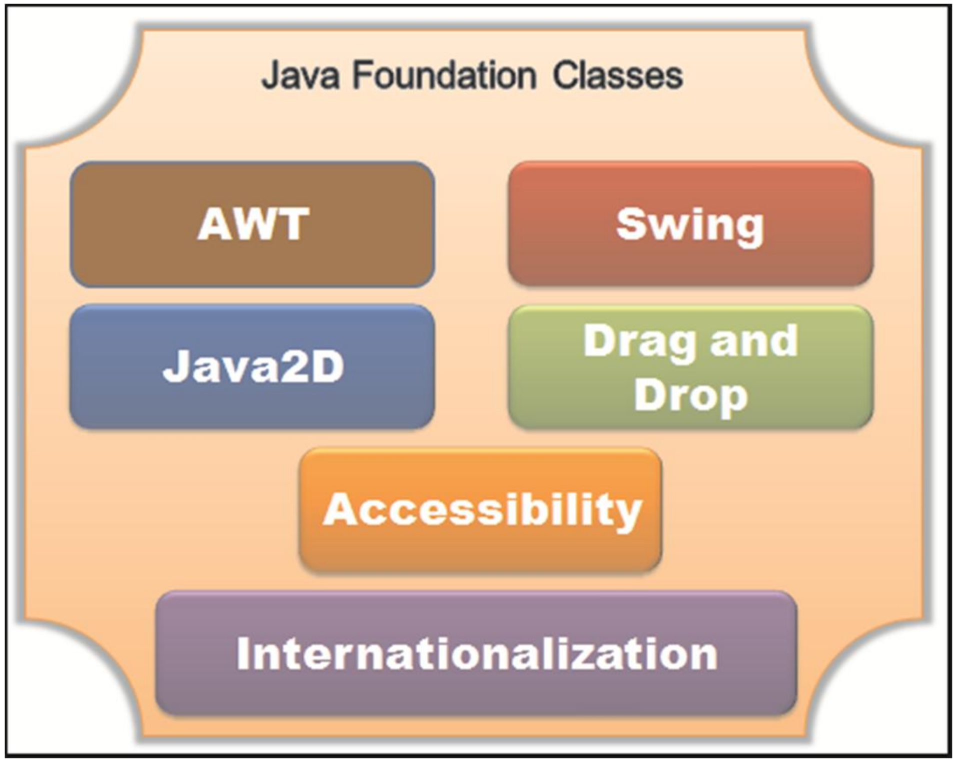


Objectives

- Explain JFC and Swing
- Explain AWT
- State the benefits of using Swing over AWT
- Explain MVC architecture
- Describe Separable Model Architecture
- Describe container and state its needs
- Describe JFrame class and its methods
- Explain JApplet class and its methods
- Describe JPanel class and its methods to create and display it
- Describe and explain various lightweight components
- Describe the steps to create Swing application in NetBeans IDE
- Explain concurrency in Swing

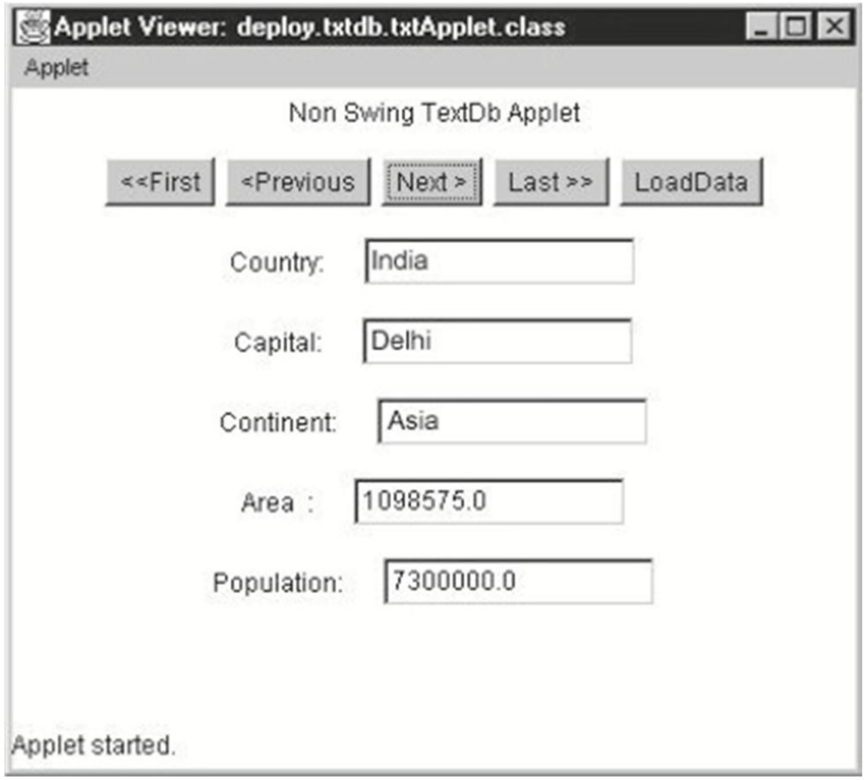
Java Foundation Classes (JFC)

- Was introduced with Java Development Kit (JDK) 2.0.
- Is a graphical framework which includes the older AWT, Swing, and Java2D classes.
- Together, they provide a consistent user interface for Java programs, regardless of whether the underlying Operating System (OS) used is Windows, Macintosh, or Linux.
- Figure displays the JFC framework.



Abstract Window Toolkit (AWT) [1-2]

- Was the initial library available to develop Graphical User Interface (GUI) applications.
- Is used to create a GUI based standalone application or applets.
- Contain the classes to represent components such as Button, CheckBox, TextField, and so on.
  - Based on peer-components that are platform specific components offered by the underlying OS.
- Figure shows the AWT components added in a Java Applet.



Country: India

Capital: Delhi

Continent: Asia

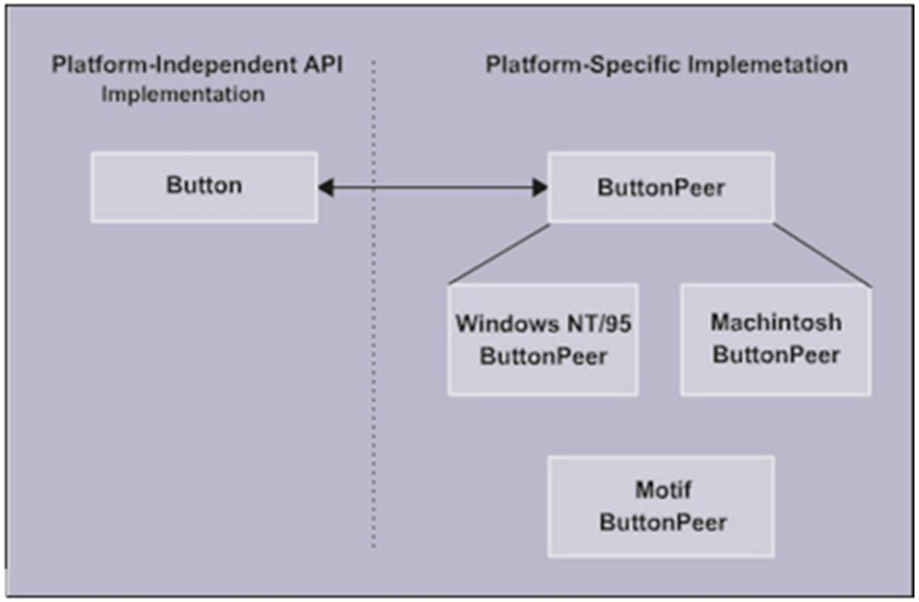
Area : 1098575.0

Population: 7300000.0

Applet started.

Abstract Window Toolkit (AWT) [2-2]

- Drawbacks of AWT are as follows:
  - Peer-component dependency made the AWT components heavy on memory consumption and operational performance.
  - Rendering of the actual component was achieved by the native-code (C/C++) and not directly by Java.
  - Figure displays the platform-specific implementation of AWT component.

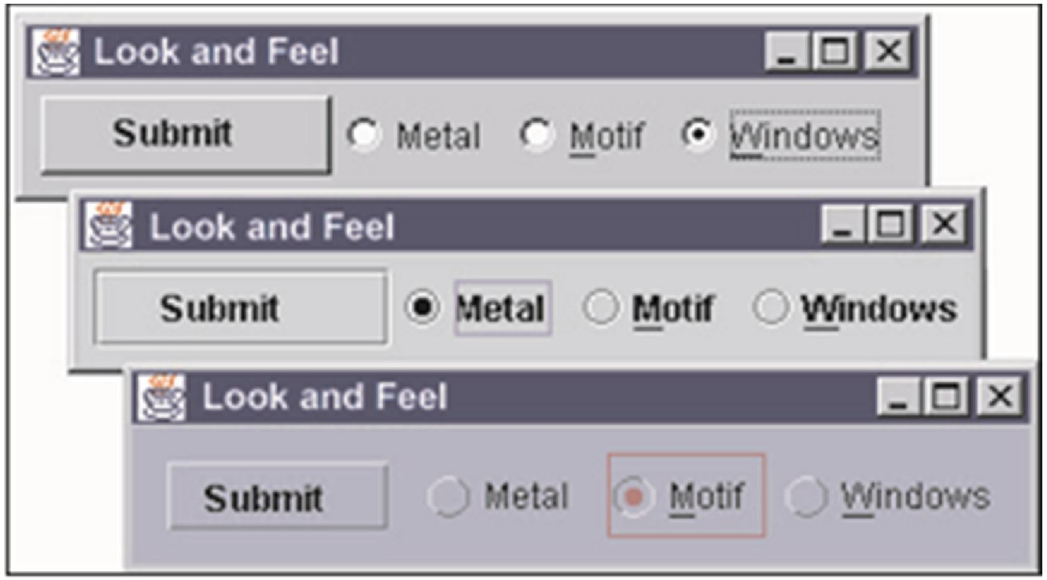


On a Linux OS, an AWT-based Java application would have a different look and feel than the one on a Windows OS.

AWT components are also referred to as heavy-weight components.

Swing

- Is a framework based on the Model-View-Controller (MVC) architecture as compared to AWT.
- Supports a technology called ‘Pluggable-Look-And-Feel’ (PLAF) where components can be displayed as on any OS.
- Example: In Linux OS, a button can be shown as it looks on a Windows OS.
- Figure displays the look and feel of Swing components on different platforms.



Swing components are also referred to as light-weight components.

Capabilities of Swing Components

Swing Buttons and Labels can display images in addition to textual labels.

The behaviour or appearance of a Swing component can be changed by either invoking methods on it or by creating a subclass of it.

Borders drawn around most of the Swing components can be changed.

Swing components do not have to be rectangular. For example, Buttons can be round.

Swing is based on the MVC architecture that offers a very streamlined way of representing and performing the components functionality.

Swing MVC Architecture [1-3]

- Is a method of separating the visual, interaction, and data components.
- Is an object-oriented UI design pattern first introduced in Smalltalk in the late 1970s.
- Decomposes the three functionality into three distinct objects:
  - Model
  - View
  - Controller

Swing MVC Architecture [2-3]

Model

- Represents all the data and the various states of the component.
- Component will be in various states because of user interaction.
- Information about these states will be maintained in the Model.
- Is responsible for indirect communication with the view and the controller.

View

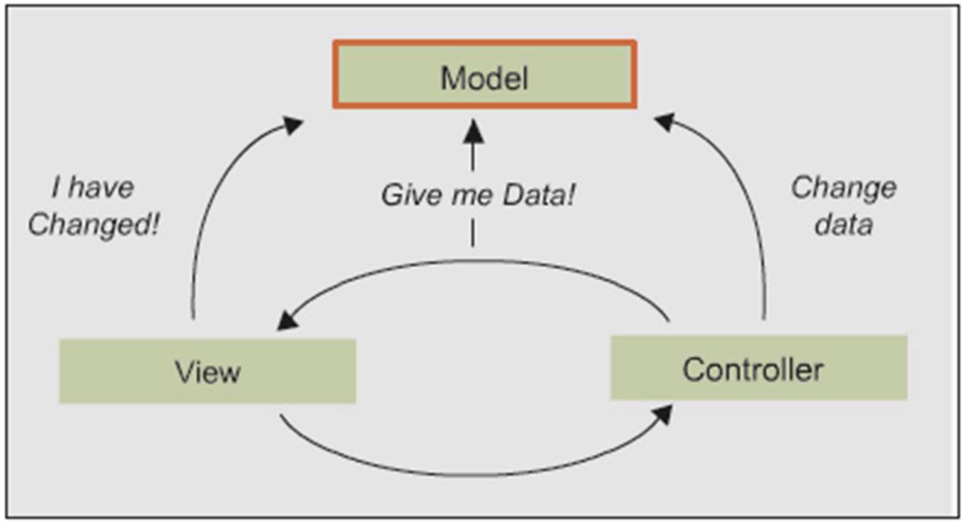
- Depicts the graphical part on the screen.
- Takes the data and various states from the Model to render the component's graphical part.
- Determines the visual representation of the component's Model.
- Is responsible for keeping its on-screen representation updated and may do so upon receiving indirect messages from the Model, or direct messages from the Controller.

Swing MVC Architecture [3-3]

Controller

- Is responsible for determining whether the component should react to any input events from input devices such as the keyboard or mouse.
- Determines what actions are performed when the component is used.
- Can receive direct messages from the View and indirect messages from the Model.

- Figure displays the MVC architecture.



Swing Separable Model Architecture

- Swing packages each component’s View and Controller into an object called as UI delegate.
- This UI delegate is responsible to propagate events generated through user interaction to the model.
- Thus, underlying architecture is more accurately referred to as Separable Model architecture or the Model-Delegate architecture.

Swing Package in Java SE 7

- Swing API provides a main package named `javax.swing`.
- `javax.swing` package:
  - Needs to be imported in the Java application.
  - Provides basic Swing components such as buttons, labels, text boxes, dialog boxes, and so on.
  - Contains various sub-packages that provide specific functionality for the components.
  - Sub-packages are as follows:
    - `javax.swing.event` - Provides classes and interfaces for handling events triggered on Swing components.
    - `javax.swing.plaf` - Provides functionality to handle look and feel of the Swing components. This package provides further sub-packages, such as `javax.swing.plaf.basic`, `javax.swing.plaf.metal`, `javax.swing.plaf.nimbus`, and so on.
    - `javax.swing.text` - Provides classes to deal with editable and noneditable components. This package provides further sub-packages such as `javax.swing.text.html` and `javax.swing.text.rtf`.
    - `javax.swing.JTable` - Provides classes and interfaces to work with the table of rows and columns.
    - `javax.swing.JTree` - Provides classes and interfaces to work with the tree structure of hierarchical data.
    - `javax.swing.filechooser` - Provides classes to work with file dialog box.
    - `javax.swing.undo` - Provides functionality such as undo or redo for editable text components.

Swing Components Look and Feel

- The default look and feel of components in JDK 1.2 was the Metal Look and Feel.
- To leverage cross-platform capabilities, JDK 1.5 introduced two new look and feel.
- **GTK+**
  - The GTK Look and Feel was introduced in Linux.
  - The GTK provides mechanism for picking up themes and making changes to UI components at runtime.
  - The class `GTKLookAndFeel` implements GTK+ Look and Feel which is based on GTK+ 2.0.
- **Synth**
  - Allows you to programmatically load an XML file containing properties of particular Swing component, such as font, color, border, and so on.
  - Allows changing Look and Feel of a Swing component at runtime.

Classes and Interfaces to Implement Look and Feel

- The `javax.swing` and `javax.swing.plaf` packages have the required classes to support the Look and Feel.
- The `javax.swing` package provides hierarchy of classes for different Look and Feel.
- **`LookAndFeel` class**:
  - Is available in the `javax.swing` package.
  - Characterizes the Look and Feel of a component.
  - Is an abstract class, so cannot be instantiated.
  - Provides a concrete implementation class `BasicLookAndFeel`.
  - The `javax.swing.plaf.metal.MetalLookAndFeel` class provides the default Look and Feel.
  - The `javax.swing.plaf.metal.MetalLookAndFeel` extends the `BasicLookAndFeel` class.
  - The `javax.swing.plaf.multi.MultiLookAndFeel` class allows multiple user interface to be associated with a component at the same time.

UIManager Class

- The `javax.swing.UIManager` class is used to set and retrieve the current Look and Feel of the GUI components.
- The `UIManager` has methods to set and get the Look and Feel.
- Table lists the `UIManager` methods.

Method	Description
`setLookAndFeel(String className)`	Sets the current Look and Feel using a Look and Feel object specified in the `className`.
`LookAndFeel getLookAndFeel()`	Returns the current default Look and Feel or null.
`String getSystemLookAndFeelClassName()`	Returns the class name of the native systems Look and Feel if it exists or the name of the default cross-platform Look and Feel.
`String getCrossPlatformLookAndFeelClassName()`	Returns the class name of the default cross-platform Look and Feel which is the Java Look and Feel.

Dependencies of Look and Feel on Operating System

- Swing provides three Look and Feel, Metal, and two Look and Feel which characterizes the Look and Feel of Windows and Motif (Unix/X).
- A Look and Feel which characterizes the Macintosh is available as a separate downloadable package.
- The Windows Look and Feel is restricted for use only on the Windows operating system for copyright reasons.

Setting and Retrieving Look and Feel

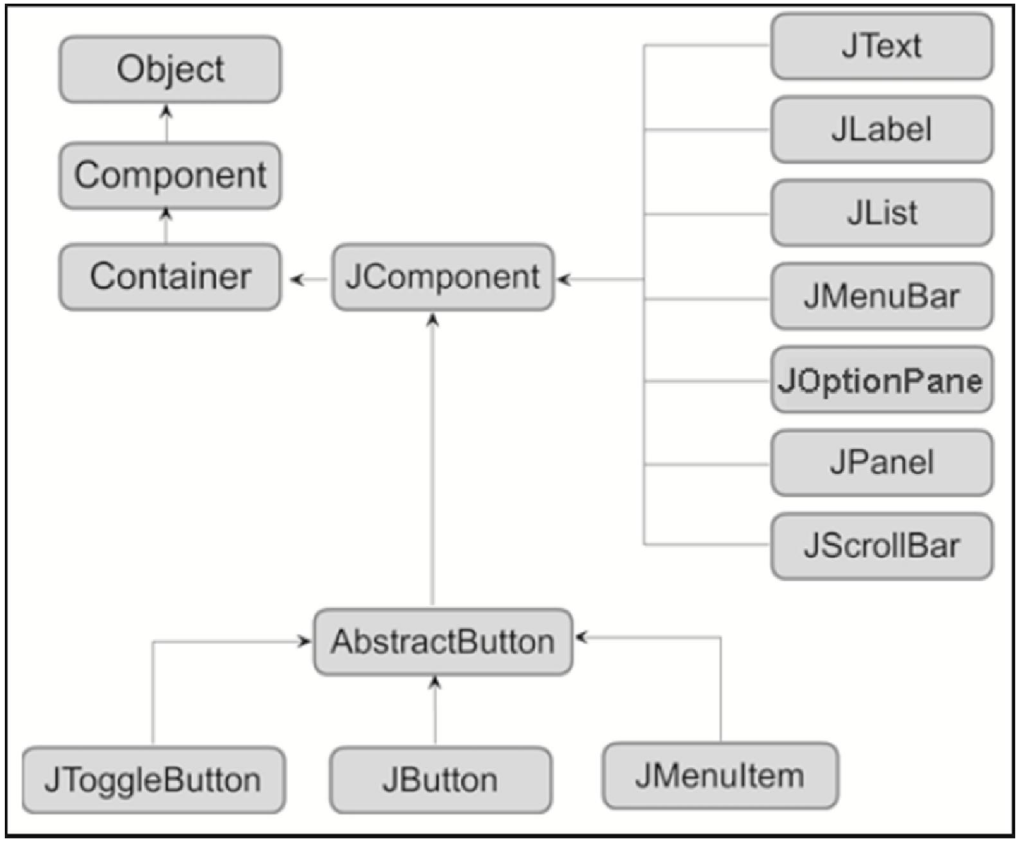
- The `setLookAndFeel()` method of the `javax.swing.UIManager` is used to set the Look and Feel.
- The Look and Feel implementation class for Windows and Motif are available in a non-standard package, and not in the core Swing package.
- The `WindowsLookAndFeel` is available in the `com.sun.java.swing.plaf.windows` package while `MotifLookAndFeel` is available in the `com.sun.java.swing.plaf.motif` package.

Code Snippet

```
try {
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.Windows
LookAndFeel");
 } catch( ClassNotFoundException ex) {
        System.out.println("Exception : " + ex.getMessage());
 } catch( UnsupportedLookAndFeelException ex){
        System.out.println("Exception : " +  ex.getMessage());
 }
```

Overview of Container [1-2]

- The `javax.swing` package provides two type of GUI components:
  - Components – Are stand-alone visual controls.
  - Containers – Are visual GUI window that is used to hold Java components.
- Figure shows the component hierarchy of `javax.swing` package.



Overview of Container [2-2]

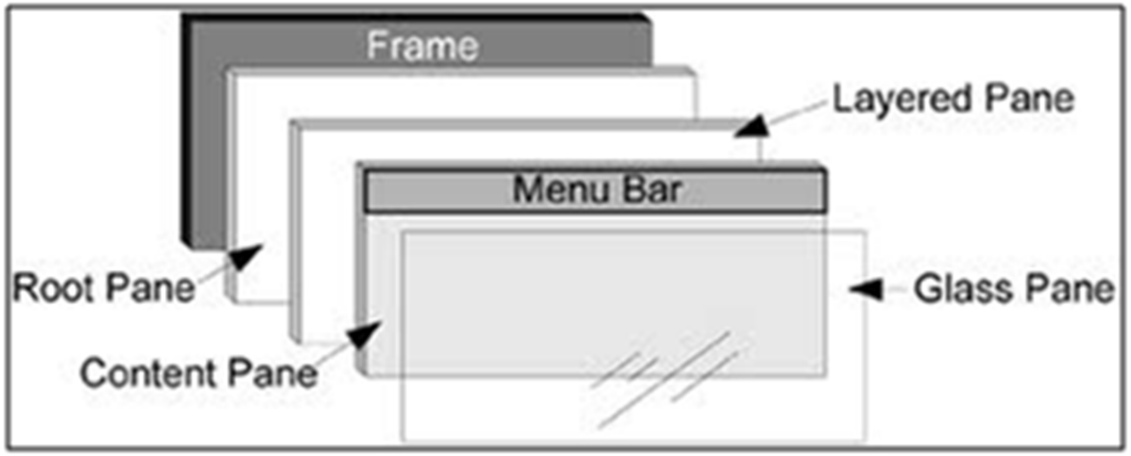
- There are three types of Containers in Swing.
- Table shows the types of containers available in Swing API.

Type of Container	Example	Description
Top Level Containers	`JApplet`, `JDialog`, `JFrame`	One of these components must be present in any swing application. They form the root of any container hierarchy.
General Purpose Containers	`JPanel`, `JScrollPane`, `JTabbedPane`, `JToolBar`	These are present in most swing applications.
Special Purpose Containers	`JInternalFrame`, `JLayeredPane`, `JRootPane`	These play specific roles in the UI.

- Each top-level container has an intermediate container called the **root pane**.

Root Pane [1-4]

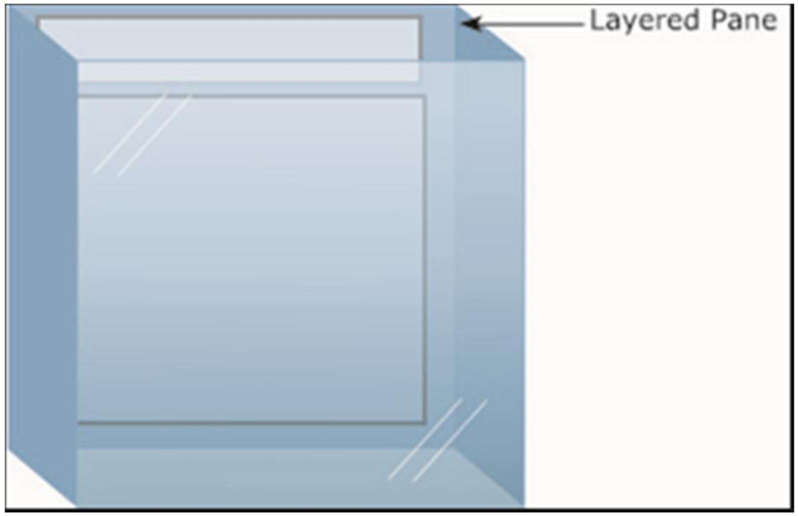
- The root pane has four parts `GlassPane`, `LayeredPane`, `ContentPane`, and `MenuBar`.



- **Glass Pane**:
  - A glass pane is hidden by default.
  - If the glass pane is made visible, then it's like a sheet of glass over all the other parts of the root pane.
  - It's completely transparent unless one implements the glass pane's `paintComponent()` method so that it displays something.

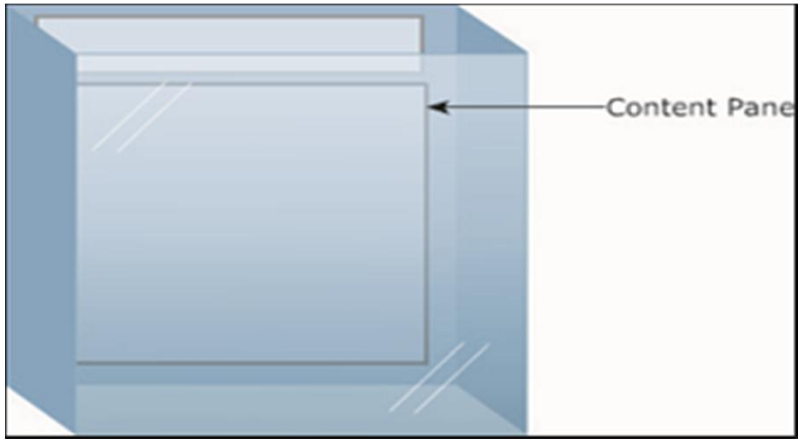
Root Pane [2-4]

- **Layered Pane:**
  - Serves to position its contents, which consist of the content pane and the optional menu bar.
  - It can also hold other components in a specified Z order.
  - Each root pane places its menu bar and content pane in an instance of `JLayeredPane`.
  - The Z ordering that the layered pane provides enables behavior such as displaying popup menus above other components.
  - Figure shows the layered pane.



Root Pane [3-4]

- **Content Pane:**
  - The `ContentPane` is the container of the root pane’s visible components.
  - Most GUI components are added to this content pane.
  - The default content pane is a simple intermediate container that inherits from `JComponent`, and uses a `BorderLayout` as its layout manager.
  - The object-reference of the content pane of a top-level container is retrieved by calling the `getContentPane()` method.
  - You then add GUI components to the content pane.
  - Figure shows the content pane.



Root Pane [4-4]

- **Menu Bar:**
  - All top-level containers can have a menu bar. In practice, however, menu bars usually appear only in `JFrame` and in `JApplet`.
  - To add a menu bar to a top-level container, a `JMenuBar` object is created, populated with menus, and then the `setJMenuBar()` method is invoked.
  - The general-purpose containers are used to add other lightweight components such as `JButton` and `JTextField`.
  - Eventually, the general-purpose containers are added to a top-level container.
  - The various general-purpose containers have different functionality depending on their use.
  - For example, a `JPanel` is an intermediate container to group components.

JFrame [1-4]

- Is a top-level container used to create a GUI-based application.
- Is available in the `javax.swing` package.
- Can be developed by using two distinct approaches:
- Extending `javax.swing.JFrame`
  - Extend `JFrame` class when the class is not extending from another class.
  - The advantages of this approach are that you can call the methods of the super class without indirection and also access the instance data of the application without specifying them as static.

JFrame [2-4]

- Code Snippet displays the use of `JFrame` class.

Code Snippet

```
import javax.swing.*;
public class WinApp extends JFrame {
    // Instance Data
             . . .
             . . .
      public WinApp() {
            . . .
      }
    public static void main(String[] args)      {
         WinApp app = new WinApp();
             . . .
    }
 }
```

JFrame [3-4]

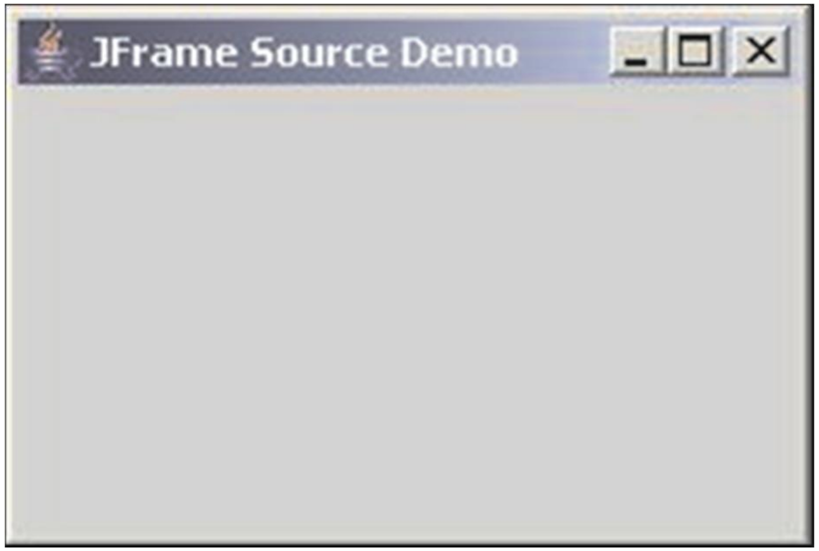
- Declaring `javax.swing.JFrame`
    - When the application is already extending another class, you cannot extend from `javax.swing.JFrame` class.
    - In such cases, an object of the `javax.swing.JFrame` class is declared in the application.
    - Code Snippet shows the creation of an object of `JFrame` class.

Code Snippet

```
import javax.swing.*;
public class WinApp extends SomeOtherClass
{
              // Instance Data       .
                . .
              public static void main(String[] args)    {
              JFrame frame = new JFrame();
                . . .
  }
}
```

JFrame [4-4]

- Figure shows a `JFrame` created by `JFrame` method.



- Components cannot be added to the `JFrame` directly. Instead they are added to the content pane of `JFrame`.

Creating a JFrame [1-2]

- Three basic things to be carried out on a frame:
  - Create a frame
  - Display a frame
  - Close the frame
- **To create a `JFrame`:**
  - A `JFrame` instance is created by using one of the following constructors:
    - `JFrame()`
    - `JFrame(String title)`
- **To display a `JFrame`:**
  - The `JFrame` after instantiation is not visible by default.
  - To make it visible, the `setVisible(boolean)` method is used which:
    - Brings the `JFrame` into a realized state and hence, no components can be added to the `JFrame` after this statement.
    - Returns `boolean` value, if set to true, then the `JFrame` is visible otherwise not.
  - A `JFrame` does not have a default initial size.
  - You have to specify its size by invoking its `setSize()` method.

Creating a JFrame [2-2]

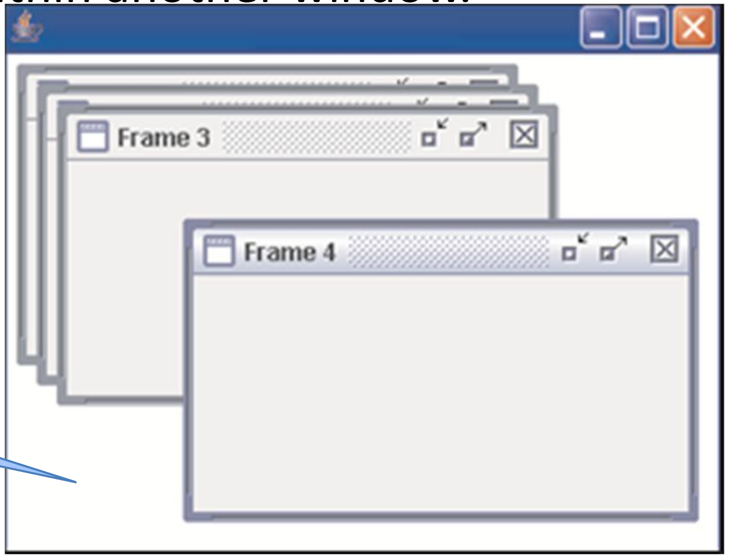
- To display a `JFrame`, you first set the size and then use the `setVisible()` method in the following manner:
  - `frmWindow.setSize(200,200);`
  - `frmWindow.setVisible(true);`
- The `pack()` method causes the window to be sized according to the preferred size and layout of its subcomponents.

- **To close a `JFrame`:**
  - The `JFrame` provides three system buttons on the top-right corner, to minimize, resize, and close.
  - The functionality of close button is partially implemented by the `JFrame` class.
  - The programmer has to either provide a default means to close the frame or register a `WindowListener` interface object to handle the close event.
  - By default, the close operation of the `JFrame` is not functional.
  - To provide the close functionality, you use the following method:

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

JInternalFrame [1-4]

- Provides the ability to create lightweight frames that exist inside other components.
- Can display a `JFrame` like window within another window.
- Allows frame operations such as dragging, closing, minimize, maximize, becoming an icon, and so on.
- Used to display a `JFrame` like window within another window.
- Figure shows an `JInternalFrame`.



Content pane of the `JInternalFrame` container where child components are added.

- The method `internalFrame.add(child)` is used to add child components.

JInternalFrame [2-4]

- Table lists the common constructors of `JInternalFrame` class.

Constructor	Description
`JInternalFrame()`	Constructs a `JInternalFrame` with no title.
`JInternalFrame(String title)`	Constructs a `JInternalFrame` with a specific title.
`JInternalFrame(String title, boolean resizable)`	Constructs a `JInternalFrame` with a title and resizability.
`JInternalFrame(String title, boolean resizable, boolean closeable)`	Constructs a `JInternalFrame` with a title, resizability, and closability.
`JInternalFrame(String title, boolean resizable, boolean closeable, boolean maximizable)`	Constructs a `JInternalFrame` with a title, resizability, closability, and maximizability.
`JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)`	Constructs a `JInternalFrame` with the specified title, resizability, closability, maximizability, and iconifiability.

JInternalFrame [3-4]

- Table lists some of the methods of `JInternalFrame`.

Method	Description
`container getContentPane()`	Returns the content pane for the `JInternalFrame`.
`void addInternalFrameListener()`	Adds a specified internal frame listener to respond to events.
`void dispose()`	This makes the internal frame invisible, unselected, and closed.
`protected void fireInternalFrameEvent()`	Fires an event of the internal frame.

- The most benefit of `JInternalFrame` is derived when added to a `JDesktopPane`.

JInternalFrame [4-4]

- Code Snippet shows how to create a desktop and internal frame.

Code Snippet

```
. . .
JDesktopPane desk1 = new JDesktopPane();
createInternalFrame();
setContentPane(desk1);
 protected void createInternalFrame()
 {
    MyInternalFrame fm1 = new MyInternalFrame();
    fm1.setVisible(true);
    desk1.add(fm1);
    . . .
    fm1.setSelected(true);
 }
. . .
```

JApplet [1-3]

- An Applet is a Java program which is meant to run as part of a Web page.
- While using Swing, the `javax.applet.JApplet` class is used to create an applet.
- The applet is embedded in the Web page with the `<applet>` tag.
- The methods of `JApplet` class are:

  `init()`

  - This method is used to initialize a `JApplet`.
  - All the one-time initialisation code is written in this method.
  - In the life cycle of a `JApplet`, this method is called only once.

  `start()`

  - This method is invoked once after the `init()` method.
  - It could be used to start threads.

JApplet [2-3]

`paint()`

- This method is called when the applet interface is to be redrawn.
- The method accepts a parameter of `java.awt.Graphics` class.
- This graphics object has several methods to draw primitive graphics elements such as line, ellipse arc, and so on.

`stop()`

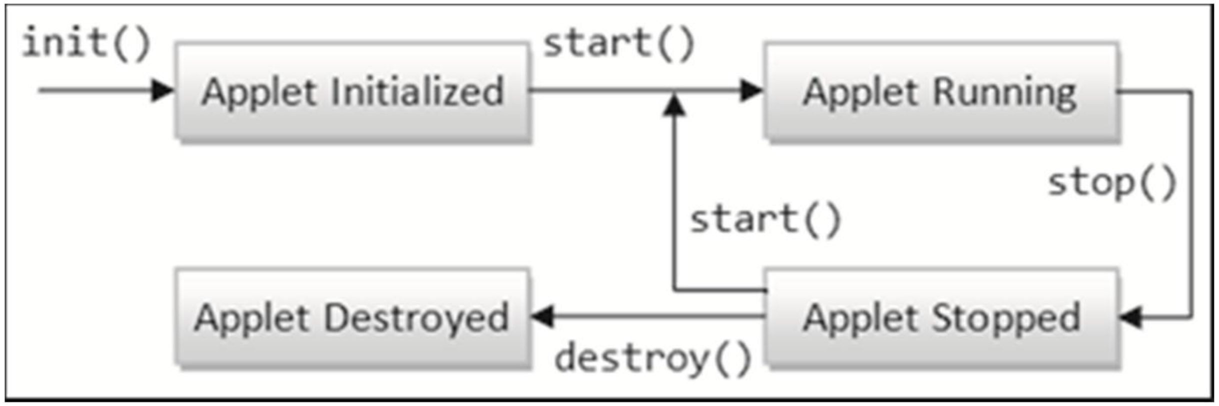
- This method is invoked when the user leaves the page that contained the applet.
- This method can be called more than once.

`destroy()`

- When the applet is unloaded from memory, this method is invoked.
- In the life-cycle of a `JApplet` it is invoked only once.
- This method is used to free up resources such as images, network, or database connections.

JApplet [3-3]

- Figure shows the method call of the `JApplet` class.



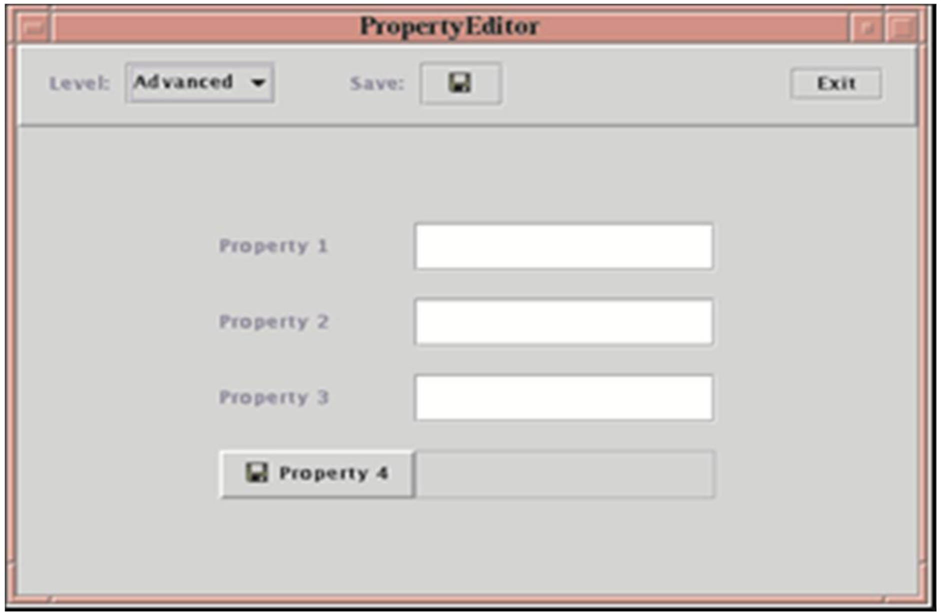
- Code Snippet creates an applet by sub classing the `JApplet` class.

Code Snippet

```
import javax.swing.JApplet;
. . .
public class MyApplet extends JApplet {
      public void init()  {  . . .   }
      public void start() {  . . .   }
}
```

JPanel [1-2]

- Is both a container and a component.
- Is added to a top-level container, or even another `JPanel` for nesting purposes.
- Is a rectangular opaque component without any visible border by default but border can be added for demarcation.
- Other components such as `JButton`, `JTextField`, and so on can be added to `JPanel`.
- Adding components to a `JPanel` first and then adding it to a top-level container is more advantageous than adding components directly.
- Figure displays a `JPanel`.



JPanel [2-2]

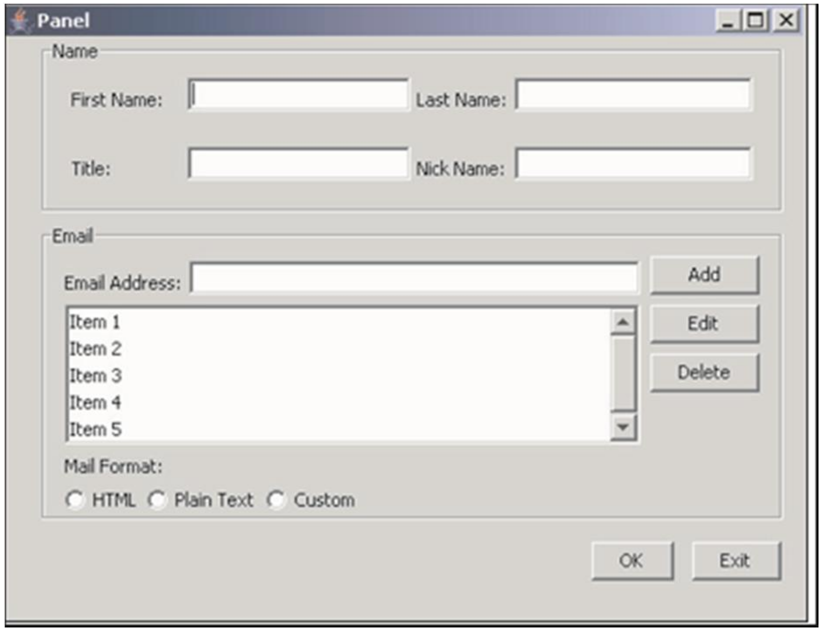
- A `JPanel` instance is created by invoking its default constructor.
- The default layout manager of `JPanel` is `FlowLayout`.
- By default, the `JPanel` as a component is visible.
- You can make it invisible by using the `setVisible(boolean)` method.
- If `boolean` parameter is set to true, `JPanel` is visible else it is not visible.

Code Snippet

```
import javax.swing.JPanel;
. . .
JPanel myPanel;
myPanel = new JPanel();
. . .
. . .
```

Lightweight Components

- Swing components are referred to as lightweight components as:
  - These components do not have their peer-components equivalent as compared to AWT components.
  - They render (draw) themselves with pure Java code.
  - They do not consume extra overheads in terms of memory and execution time.
  - They are platform independent.
  - They have 'Pluggable-Look-And-Feel' capabilities that allow to virtually copy the look and feel of any operating system on other operating systems.
  - Figure displays the lightweight components.



JLabel [1-2]

- `JLabel`:
  - The `JLabel` is a component to display static text in a GUI-based application.
  - A `JLabel` can also display an icon, or both text and icon.
  - Table lists the `JLabel` constructors.

Constructor	Description
`JLabel()`	Creates a label without text.
`JLabel(String label)`	Creates a label with the specified text; where `label` is a string.
`JLabel(ImageIcon icon)`	Creates a label with the specified image; where `icon` is an image.
`JLabel(String label, ImageIcon icon)`	Creates a label with the specified text and image; where `label` is a string and `icon` is an image.

  - The two important methods are `setText(String label)` and `getText()`.

JLabel [2-2]

- Code Snippet shows how to create a label with a caption and an image. It also shows how to retrieve its caption.

Code Snippet

```
// Import the JLabel and ImageIcon class
import javax.swing.JLabel;
import javax.swing.ImageIcon;
. . .
// Create an image
    ImageIcon icon = new ImageIcon(“/images/name.gif”);
// Create a label with an image in it
    JLabel lblName = new JLabel(“Name”, icon);
// Add the label to a container
   getContentPane().add(lblName);
. . .
// Retrieve the caption of lblName component
   String caption = lblName.getText();
. . .
```

JButton [1-2]

- `JButton`:
  - Is a rectangular component with a text or icon as its label, and can fire an event when clicked with a mouse.
  - Table lists the constructors of the `JButton`.

Constructor	Description
`JButton()`	Creates a button without a label.
`JButton(String label)`	Creates a button with the specified label; where `label` is a string.
`JButton(ImageIcon icon)`	Creates a button with the specified image; where `icon` is an image.
`JButton(String label, ImageIcon icon)`	Creates a button with the specified label and image; where `label` is a string and `icon` is an image.

  - The two important methods are `setText(String label)` and `getText()`.
  - Example: `JButton btnOn = new JButton(“On”);`
    `btnOn.setText(“Off”);`

JButton [2-2]

- Code Snippet shows how to create a button containing a caption and an image, and add it to the container.

Code Snippet

```
// Import the JButton and ImageIcon class import javax.swing.JButton;
    import javax.swing.ImageIcon;
. . .
// Create an image
    ImageIcon icon = new ImageIcon("/images/ok.gif");
// Create a button with an image on it
    JButton lblName = new JButton("Name", icon);
// Add the button to a container
   getContentPane().add(lblName);
. . .
```

- A `JButton` label can consist of:
  - Multiple lines of text.
  - Use html text in the string label.
  - Example:

```
JButton btnRightJustify = new JButton
("<html><b><i>Right<br>Justify</b></i> </html>");
```

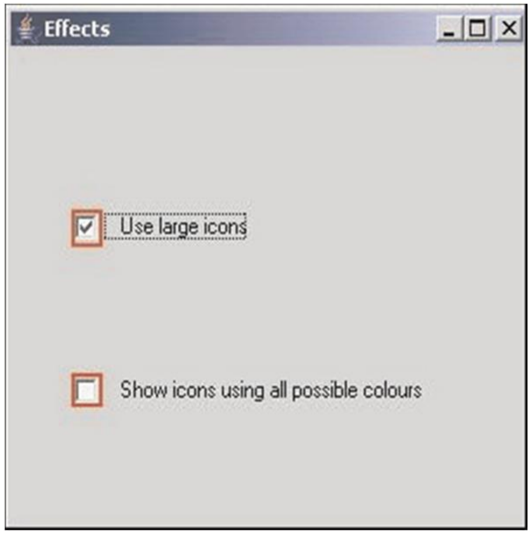
JCheckBox

- `JCheckBox`:
  - Creates a component which can have only two states, either checked or unchecked. By default, unchecked.

1. Some of the constructors of the `JCheckBox`:
   - `JCheckBox()`
   - `JCheckBox(String label)`
   - `JCheckBox(String label, boolean state)`
   - `JCheckBox(ImageIcon icon)`
   - `JCheckBox(String label, ImageIcon icon)`
   - Example: To create the `JCheckBox` labelled 'Bold'.

```
 JCheckBox chkBold;
chkBold = new JCheckBox("Bold");
```

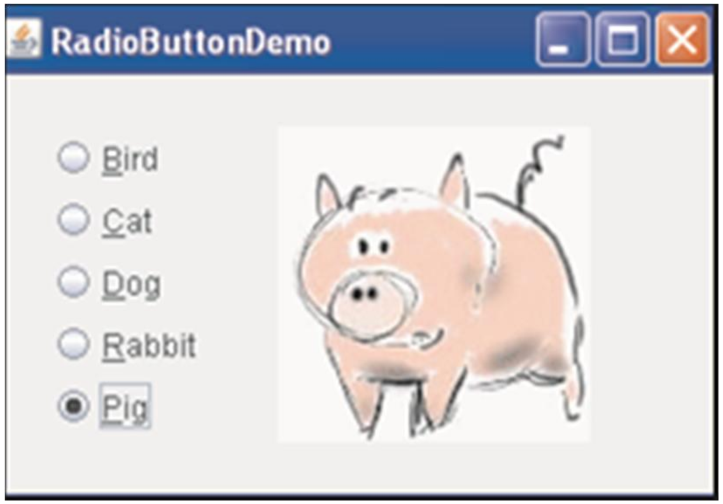
- The two important methods are `boolean isSelected()` and `void setSelected(boolean state)`.
  - Example: `chkBold.setSelected(true);`



JRadioButton

- `JRadioButton:`
  - Creates a component which can have only two states, either checked or unchecked, by default it is unchecked.
  - Constructors of the `JRadioButton` are:
    - `JRadioButton()`
    - `JRadioButton(String label)`
    - `JRadioButton(String label, boolean state)`
    - `JRadioButton(ImageIcon icon)`
    - `JRadioButton(String label, ImageIcon icon)`
  - Example:

```
JRadioButton radBold = new
JRadioButton(“Bold”);
```



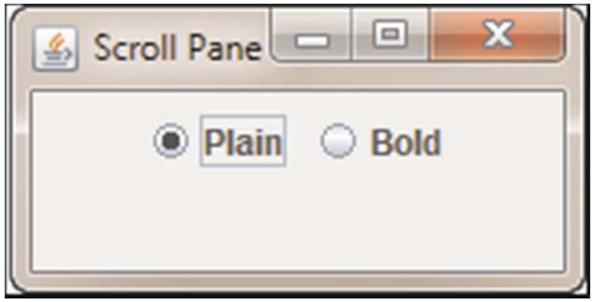
Grouping JRadioButton

- Provides a convenient way to perform multiple selections of choices and allows user to select only one of all possible choices.
- Provides a class `javax.swing.ButtonGroup` which is used to group radio buttons which fall in the same category.
- Code Snippet shows how to add `JRadioButton` to a group.

Code Snippet

```
. . .
JRadioButton radPlain, radBold;
ButtonGroup group;
radPlain = new JRadioButton(“Plain”, true);
container.add(radPlain);
radBold = new JRadioButton(“Bold”);
container.add(radBold);
group = new ButtonGroup();
group.add(radPlain);
group.add(radBold);
. . .
```

- Output:



JTextField

- Allows to input and edit a single line of text.
- Normally text is inputted by the user, however you can also set text in a `JTextField` programmatically.
- Has the following constructor methods:
  - `JTextField()`
  - `JTextField(String initialText)`
  - `JTextField(String initialText, int columns)`
- Example: `JTextField txtMessage = new JTextField();`
- The methods of `JTextField` are: `getText()`, `setText(String text)`, `setEditable(boolean editable)`.
- Code Snippet shows how to create, display, and set the text as a non-editable text field.

Code Snippet

```
. . .
JTextField txtMessage;
txtMessage = new JTextField();
container.add(txtMessage);
txtMessage.setText("Enter correct password");
txtMessage.setEditable(false);
. . .
```

JTextArea [1-2]

- Allows to input and edit multiple lines of text.
- Normally text is entered by the user; however you can also set text in a `JTextArea` programmatically.
- Has the following constructor methods:
  - `public JTextArea()`
  - `public JTextArea(String text)`
  - `public JTextArea(int rows, int columns)`
- Example: `JTextArea txaInfo = new JTextArea(4,10);`
- Other important methods of `JTextArea` are:
  - `copy()` and `cut()` - Transfers the currently selected text from the text area to the system clipboard.
  - `paste()` - Transfers the contents of the system clipboard into the text area at the current caret position.

JTextArea [2-2]

- Code Snippet demonstrates the use of `copy()` method.

Code Snippet

```
. . .
JTextArea jtaNotes;
String strSelectedText;
 . . .
strSelectedText = jtaNotes.getSelectedText();
 if (strSelectedText.equals(""))
 {
      jtaNotes.copy();
 }
strSelectedText = jtaNotes.getSelectedText();
 if (strSelectedText.equals(""))
 {  jtaNotes.cut(); }

btnPaste.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
    {  jtaNotes.paste(); }
});
```

JPasswordField

- Is similar to a `JTextField` in appearance.
- A asterisk character (`*`) is echoed for every character typed in the field.
- By default, the asterisk character is echoed, it can be programmatically changed to any other character desired.
- Example: `JPasswordField pwdPass = new JPasswordField();`
- Some of the methods of `JPasswordField` are `getText()`, `setText()`, and `setEchoChar()`.
- Code Snippet shows how to create a `JPasswordField`.

Code Snippet

```
. . .
JPasswordField pwdPass;
pwdPass = new JPasswordField();
container.add(pwdPass);
String strPassword = pwdPass.getText();
. . .
```

Event Handling [1-2]

- An event is occurred when a user interacts with the GUI component.
- Delegation event model is used for handling events.
- Components of delegation event model are as follows:

Event

- It is an object describing a state change that has occurred in the source such as pressing of a button or selection of an item in a list.
- An event can also be generated where there is no direct interaction with the user such as expiry of a timer.

Event Source

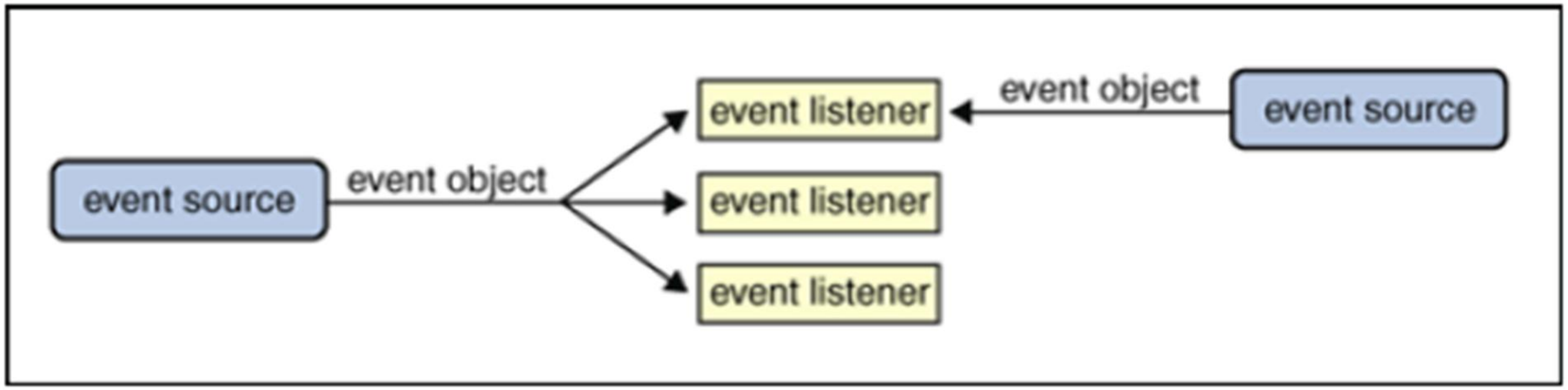
- It is an object generating the event. Source can generate more than one type of event.

Listeners

- It is an object that receives notification when an event occurs. The listeners must be registered with the source to receive the notification.

Event Handling [2-2]

- Figure shows the components of delegation event model.



- **Registering the Events**
  - An object that generates events can perform registering for its event handlers.
  - The event handlers are the listeners that must be registered with the source, before receiving any event notification.
  - The benefit of this is that when events are raised, notifications are sent only to registered listeners who can handle the event.

Event Listeners and their Methods [1-3]

- The `java.awt.event` package defines different types of events generated by GUI components in Swing.
- Some of the events which are specific to Swing are defined in `javax.swing.event` package.
- Table lists some of the events.

Events	Source	Listeners	Interface Methods
`ActionEvent`	Button, Menu, List	`ActionListener`	`void actionPerformed(ActionEvent ae)`
`AdjustmentEvent`	Scrollbar	`AdjustmentListener`	`void adjustmentValueChanged(AdjustmentEvent ae)`
`ItemEvent`	Check box, List	`ItemListener`	`void itemStateChanged (ItemEvent ie)`

Event Listeners and their Methods [2-3]

Events	Source	Listeners	Interface Methods
MouseEvent	Mouse	MouseListener MouseMotionListener	void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me) void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me) void mouseDragged(MouseEvent me) void mouseMoved(MouseEvent me)

Event Listeners and their Methods [3-3]

Events	Source	Listeners	Interface Methods
WindowEvent	Window	WindowListener	void windowActivated(WindowEvent we) void windowClosed(WindowEvent we) void windowClosing(WindowEvent we) void windowDeactivated(WindowEvent we) void windowDeiconified(WindowEvent we) void windowIconified(WindowEvent we) void windowOpened(WindowEvent we)

Event Handling for JButton [1-2]

- The `JButton` component listens using the `java.awt.event.ActionListener` interface which:
  - Has one method named `actionPerformed()`.
  - The application class can implement the `ActionListener` interface and provide the implementation of `actionPerformed()` method.
- Finally, button need to be registered with the method `addActionListener()` to receive events.
- The implementation of `actionPerformed()` method can be done in two ways:
  - Use an Inner-class as the event-handler.
  - Use an Anonymous-class as the event-handler.

Event Handling for JButton [2-2]

- Code Snippet shows how to handle the event of JButton labeled **'Ok'** with the help of anonymous class.

Code Snippet

```
. . .
JButton btnOk;
btnOk = new JButton("Ok");
container.add(btnOk);
. . .
 // Anonymous Class
   btnOk.addActionListener(new ActionListener()
   {
       public void actionPerformed(ActionEvent e)
       {
         // Action Code
            . . .
       }
   });
```

Event Handling for JCheckBox

- A `JCheckBox` component listens using either `java.awt.event.ActionListener` or `java.awt.event.ItemListener` interface.
    - The `java.awt.event.ItemListener` interface has one method namely, `public void itemStateChanged(ItemEvent e)`
- A `JCheckBox` has the two methods for registering listener objects namely, `addActionListener()` and `addItemListener()`.
- Code Snippet shows how to handle the event of `JCheckBox` labeled '**`Bold`**'.

Code Snippet

```
JCheckBox chkBold;
chkBold = new JCheckBox ("Bold");
container.add(chkBold);
. . .
// Anonymous Class
   chkBold.addItemListener(new ItemListener()
   {
    public void itemStateChanged(ItemEvent e)
    {
       // Action Code
          . . .
    }
  });
```

Event Handling for JRadioButton [1-2]

- A JRadioButton component listens using either java.awt.event.ActionListener or java.awt.Event.ItemListenerinterface.
- A JRadioButton has two methods for registering listener-objects namely, addActionListener() and addItemListener().
- Code Snippet shows how to create a JRadioButton.

Code Snippet

```
//Create the radio buttons.
JRadioButton birdButton = new JRadioButton(birdString);
birdButton.setMnemonic(KeyEvent.VK_B);
birdButton.setActionCommand(birdString);
birdButton.setSelected(true);


JRadioButton catButton = new JRadioButton(catString);
catButton.setMnemonic(KeyEvent.VK_C);
catButton.setActionCommand(catString);
```

Event Handling for JRadioButton [2-2]

```
JRadioButton dogButton = new JRadioButton(dogString);
dogButton.setMnemonic(KeyEvent.VK_D);
dogButton.setActionCommand(dogString);

JRadioButton rabbitButton = new JRadioButton(rabbitString);
rabbitButton.setMnemonic(KeyEvent.VK_R);
rabbitButton.setActionCommand(rabbitString);

JRadioButton pigButton = new JRadioButton(pigString);
pigButton.setMnemonic(KeyEvent.VK_P);
pigButton.setActionCommand(pigString);

//Group the radio buttons.
...
//Register a listener for the radio buttons.
   rabbitButton.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
       picture.setIcon(new ImageIcon("images/" e.getActionCommand()
        + ".gif"));
    }
   });
```

Event Handling for JTextField

- A `JTextField` listens using two listener interfaces:
  - `ActionListener` – Registered using `addActionLister().`
  - `FocusListener` – Registered using `addFocusListener().`
- `FocusListener` has methods:
  - `public void focusGained(FocusEvent e)` - Is invoked when component gains focus.
  - `public void FocusLost(FocusEvent e)` - Is invoked when component loses focus.
- Code Snippet shows how to handle focus events of the `JTextField`.

Code Snippet

```
JTextField txtName;
txtName = new JTextField();
container.add(txtName);
. . .
txtName.addFocusListener(new FocusListener()
{
     public void focusGained(FocusEvent e)
        { . . . }
     public void focusLost(FocusEvent e)
        { . . . }
});
```

Event Handling for JTextArea

- A JTextArea listens for events using java.awt.event.ActionListener interface.
- Code Snippet shows how to handle the action event of the JTextArea.

Code Snippet

```
JTextArea txaNotes;
txaNotes = new JTextField();
container.add(txaNotes);
txaNotes.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
       // Code to handle Enter key
           . . .
           . . .
  }
});
```

Event Handling for JPassword

- A `JPasswordField` listens for events using `java.awt.event.ActionListener` interface.
- Code Snippet shows how to add event handler to the `JPasswordField`.

Code Snippet

```
JPasswordField pwdPass;
pwdPass = new JPasswordField();
container.add(pwdPass);
pwdPass.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
     // Code to handle Enter key
         . . .
   }
});
```

KeyListener [1-2]

- Handles key events such as:
  - `KeyPressed (Key Event)` – Is fired when any key is pressed.
  - `KeyReleased (Key Event)` – Is fired when any key is released.
  - `KeyTyped (Key Event)` – Is fired when something is typed from the keyboard.
- The key event is represented by `KeyEvent` object which has following methods:
  - `char getKeyChar()`
  - `int getKeyCode()`
  - `public int getKeyLocation()`
  - `public getKeyText(int keyCode)`
  - `public boolean isActionKey()`
  - `void setKeyChar(char keyChar)`
  - `void setKeyCode(int keyCode)`
- Code Snippet demonstrates the use of `keyListener`.

Code Snippet

```
public class MyPanel extends JPanel implements KeyListener {
private char c = 'e';
```

KeyListener [2-2]

```
public MyPanel() {
   this.setPreferredSize(new Dimension(500,500));
  addKeyListener(this);
 }
 public void paintComponent(Graphics g) {
   super.repaint();
   g.drawString("the key that pressed is" + c, 250,250);
 }
 public void keyPressed(KeyEvent e) {
     c=e.getKeyChar();
     repaint();
 }
 public void keyReleased(KeyEvent e) {
 }
      public void keyTyped(KeyEvent e) {
      c=e.getKeyChar();
      repaint();
 }
  public static void main(String[] s) {
    JFrame f=new JFrame();
    f.getContentPane().add(new MyPanel());
    f.pack();
    f.setVisible(true);
}}
```

FocusListener [1-2]

- Focus events are fired whenever a component obtains keyboard focus or loses keyboard focus.
- Whenever a component obtains keyboard focus, `focusGained()` method is invoked and whenever a component loses keyboard focus, `focusLost()` method is invoked.
- The `addFocusListener()` method is used to register the class with the component.
- `public void addFocusListener(FocusListener e)`
- When the focus event occurs (focus gained or lost by a component), a focus event is generated which will pass the `FocusEvent` to the relevant method in the listener object.

FocusListener [2-2]

- Code Snippet demonstrates how to implement a `FocusListener`.

Code Snippet

```
exTextArea = new JTextArea("Event Feedback..");
exTextArea.addFocusListener(this);
JScrollPane scrollText = new JScrollPane(exTextArea);

// The FocusListener methods append information about
// the focus events to the JTextArea
@Override
public void focusGained(FocusEvent e) {
exTextArea.append(e.getComponent().getClass().getName() + " has
                 got the focus\n");
}
@Override
 public void focusLost(FocusEvent e) {
  exTextArea.append(e.getComponent().getClass().getName() + " has
                 lost the focus\n");
}
```

MouseListener [1-2]

- To handle mouse events, the `MouseListener` and the `MouseMotionListener` interfaces are implemented.
- Mouse events take place whenever the user presses or releases a mouse or enters and leaves a component’s onscreen area.
- The `addMouseListener()` method can be used to add a mouse listener to the components to receive mouse events.
- `public void addMouseListener(MouseListener e)`
- Code Snippet demonstrates the use of `MouseListener` interface.

Code Snippet

```
JTextArea jtaNotes;
. . .
 jtaNotes.addMouseListener(new MouseListener() {
      public void mousePressed( MouseEvent e)
      {
        // Add code to handle mouse button pressed
          . . .
      }
```

MouseListener [2-2]

```
 public void mouseReleased( MouseEvent e)
 {
    // Add code to handle mouse button released
       . . .
 }
public void mouseClicked( MouseEvent e)
 {
           // This method is invoked as a result of
           // mouse button pressed and released
            . . .
 }
public void mouseEntered( MouseEvent e)
 {
                    // This method is invoked when the mouse
                   // pointer enters into the textarea
 }
public void mouseExited( MouseEvent e)
 {
                    // This method is invoked when the mouse
                   // pointer exits the textarea
 }
 });
```

Setting Up Project Using NetBeans IDE

- To create a swing project using NetBeans IDE, perform the following steps:
  - To add a new project to the NetBeans IDE, click **File** menu and select **New Project** to open the **New Project** dialog box.
  - To select Java in the **Categories pane**, click **Java Application** under **Projects pane**.
  - Click **Next**.
  - Enter **project1** in the **Project Name** box to provide a name to the project. Select the location where you want the project to be created.
  - In the **Create Main Class** box, delete default content.
  - Select **Set as Main Project** check box.
  - Click **Finish**.

Developing GUI Application in NetBeans IDE [1-2]

- To add a blank form, perform the following steps:
  - Right-click the project name and click **New** → **JFrame Form**.
  - Enter **Form1** in the **Class Name** box.
  - Enter **formpack** in the **Package** box.
  - Click **Finish** to create the form.
- **Add Components**:
  - Select the panel component in the **Palette** window, by clicking and releasing the mouse button.
  - By moving the cursor to the upper left corner of the GUI Builder form, locate the component near the top left edges of the container.
  - Align the component according to the horizontal and vertical guidelines that appear.
  - Click in the form to place the components in this location.

Developing GUI Application in NetBeans IDE [2-2]

- **Align Components**:
  - Press the **Ctrl** key and click to select the component on the left side of the form.
  - Click the **Align Right** in **Column** button in the toolbar. You can also right-click either one and choose **Align** → **Right** in **Column** from the popup menu.
  - Repeat this for every component that needs aligning.
- **Finish Up**:
  - Select the **Button** component in the **Palette** window.
  - Move cursor over the form until guideline appears to indicate the `JButton` right edge is aligned.
  - Click to place the button.
  - Set display text for the button.
  - Press **Ctrl+Save** to save the file.

Developing Application Logic

- **Change the Default Variable Names**:
  - Change the default names of the variables to something more meaningful as the default names are not relevant.
- **Register the Event Listeners**:
  - The NetBeans IDE makes event listener registration extremely simple.
- **Add the Code**:
  - The final step is to simply write the code into the empty method body.
  - In the **Design Area**, click **Action** button to select it.
  - Right-click the button and choose **Events** → **Action** → **ActionPerformed**. This will generate the required event-handling code, leaving you with empty method bodies in which you add your own functionality.

Compiling and Executing GUI Application

- **Build the Whole Project**:
    - Select the project node in the **Projects** window and choose **Build Project**, to compile your whole project.
    - The **Output** window allows you to view the build progress.
- If the output concludes with the statement, BUILD FAILED, you may need to check the code for syntax errors.
    - Errors are reported in the **Output** window as hyperlinked text.
    - Click a hyperlink that reports the error to navigate to the source of an error.
    - Choose **Build Project** after fixing the errors.

Common Methods of Swing Components [1-2]

- `setIcon():` The `setIcon()` method is defined in the class `javax.swing.AbstractButton.`
- `setMnemonic():` Mnemonics allows one character in the components label to be underlined. The component can be either clicked with a mouse or alternately with the **Alt** key and the mnemonic character combination.
- `setToolTipText():` A Tool Tip is a visual textual feedback from a component when the mouse cursor hovers over a component. It is especially useful when components have small icons but may not depict the immediate meaning of the component.
- `setBackground():` The `setBackground()` method is used to set the background color of the component.
- `setForeground():` The `setForeground()` method is used to set the foreground color of the component.

Common Methods of Swing Components [2-2]

- `setBorder():` Swing supports decorative and non-decorative borders, which any component inherited from `JComponent` can have. Normally these borders are useful for general-purpose containers such as `Jpanel`, `JScrollPane`, and so on which do not have any visible border; however you can set a border for any component using the `setBorder()` method.
- `setEnabled():` A component which is enabled can respond to user inputs. The components which are disabled cannot respond to user inputs. To enable a component, you use the `setEnabled()` method. All components which extends from `javax.swing.JComponent` inherit this method.
- `setFont():` The `setFont()` method is used to set the current font of the component. It accepts a `java.awt.Font` object as a parameter.
- `setVisible():` The `setVisible()` method is used to make a component either visible or invisible.

Threads in Swing

A Swing program deals with following types of threads:

- **Initial Thread**:
  - The initial thread is referred to as the main-thread.
  - When a Java program is executed, the JVM starts the main thread, which executes the `static main()` method of the application.
  - This thread is responsible for all aspect of the program's execution including memory access, I/O operations and so on except the Swing related GUI.
- **Event-Dispatch Thread**:
  - When JVM encounters Swing related operation for the first time it starts another thread to handle the Swing components.
  - This thread is referred to as the event-dispatch thread, which is responsible for all aspects of the GUI components.
  - Swing event-handling and painting related code is placed inside event-dispatch thread.
  - Swing components and models are to be created and modified in event-dispatch thread to avoid deadlock situation.

SwingUtilities.invokeLater() Method

- The `javax.swing.SwingUtilities` class provides a static utility method, `invokeLater()` that allows the execution of arbitrary code from the event-dispatch thread.
- A `Runnable` object is passed as an argument to the `invokeLater()` method.
- The `run()` method of this object is invoked from the event-dispatch thread.
- The `invokeLater()` method:
  - Returns immediately regardless of when the `run()` method is invoked.
  - Do not run the `Runnable` object immediately.
  - Encapsulates the `Runnable` object within a special event object and places the event on the event queue.
- Using the `SwingUtilities.invokeLater()` method ensures that the imminent clash between the main-thread and the event-dispatch thread is averted.

Summary [1-2]

- JFC is a graphical framework which includes the older AWT, Swing, and Java2D. Swing is framework based on the Model-View-Controller architecture as compared to AWT which was just a toolkit.
- Model-View-Controller Architecture (MVC) is an object-oriented user interface design pattern first introduced in Smalltalk somewhere in the late 1970s.
- A Container is meant to contain components. There are two types of Containers, Top-Level Containers, and General-Purpose Containers.
- Swing components are referred to as Lightweight components. Lightweight components render (draw) themselves with pure Java code, and hence, do not depend upon external native language code such as C/C++ which forms the peer equivalent.
- NetBeans IDE provides a smooth environment for swing applications. They allow a better plumbing that the user would have to code in. NetBeans does not add overhead thus making the supplication time and work efficient.

Summary [2-2]

- Swing provides various methods that are common to most GUI components. Some of these methods are setIcon(), setMnemonic(), setToolTipText(), setBackground(), setForeground(), setBorder(), setEnabled(), setFont(), and setVisible() method.
- A good swing application uses concurrency to ensure it never freezes. This is handled by Swing Threads such as Event-Dispatch Thread and Initial Thread.