Distributed Programming in Java

Session: 2

Basic Swing Components

Objectives

- Explain JScrollPane and its methods
- Explain JSlider and its methods
- Explain event handling of JSlider
- Describe JProgressBar and its methods
- Explain event handling of JProgressBar
- Describe JFormattedTextField
- Describe JEditorPane and JTextPane
- Explain ImageIcon and Border API
- Identify and explain the need of dialog boxes
- Identify and explain the different types of JOptionPane
- Explain JDialog and its methods

Introduction

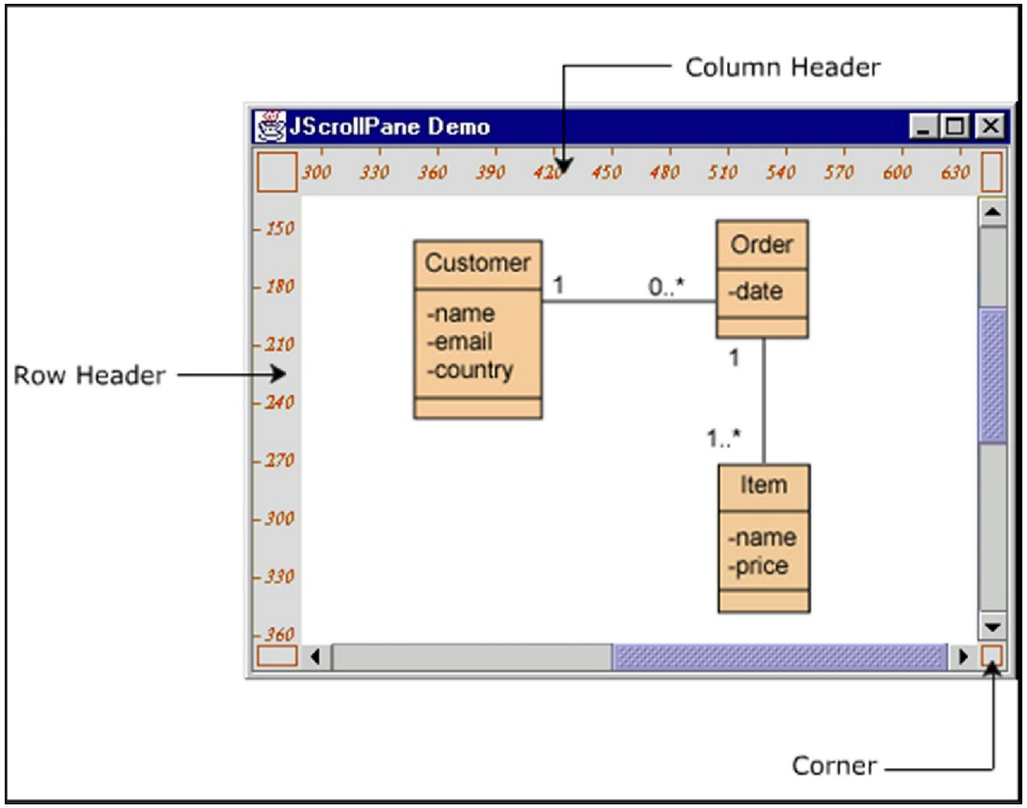
- The Swing API provides various lightweight components that are useful in visually depicting and manipulating progress.
- Some of these components are as follows:
  - `JScrollPane` – Allows scrolling of a component’s view.
  - `JSlider` - Allows the user to select a value by sliding a knob at a given interval.
  - `JProgressBar` - Allows the user to follow the progress of a task.

JScrollPane [1-4]

- A `JScrollPane` provides a scrollable view of a component.
- Some of the Swing components such as `JTextArea`, `JList`, `JTable`, and `JTree` require more space than allocated initially, because they expand.
- A `JScrollPane` can scroll any component inherited from a `JComponent`.
- A `JScrollPane` provides both horizontal and vertical scrolling.
- The `JScrollPane` as part of its view provides four corners where you can add components. These corners are fixed and do not scroll with the scroll bars.
- Apart from these four corners, two more headers are provided namely, Row Header and Column Header.

JScrollPane [2-4]

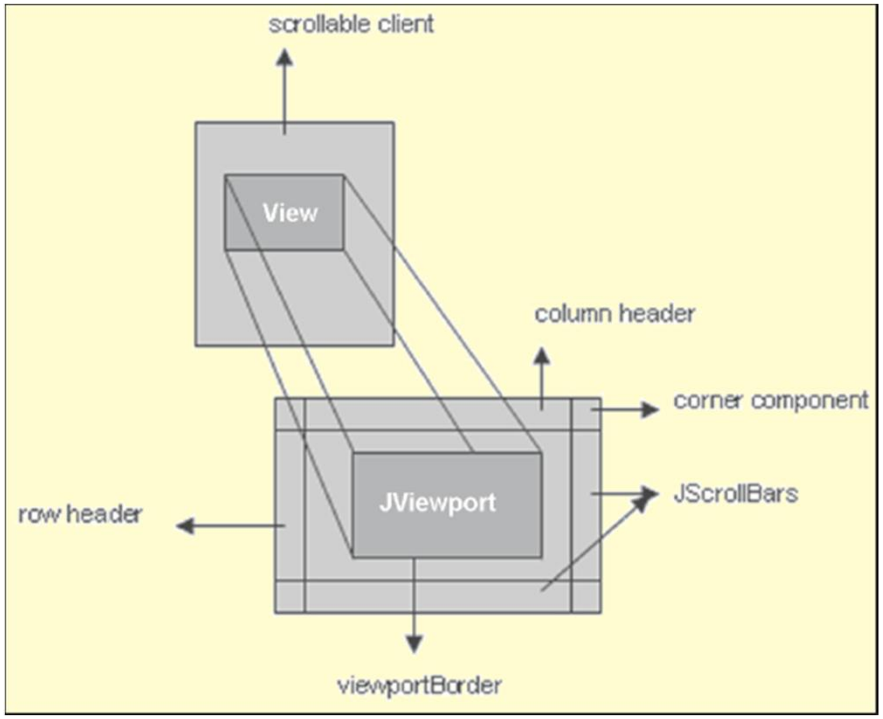
- Figure shows the corners and headers of `JScrollPane`.



  - **`RowHeader`** – A row header is provided between the top-left corner and the bottom-left corner, which can scroll vertically as the scroll bars start scrolling.
  - **`ColumnHeader`** – A column header is provided between the top-left and top-right corners, which scroll as the scroll bars start scrolling.

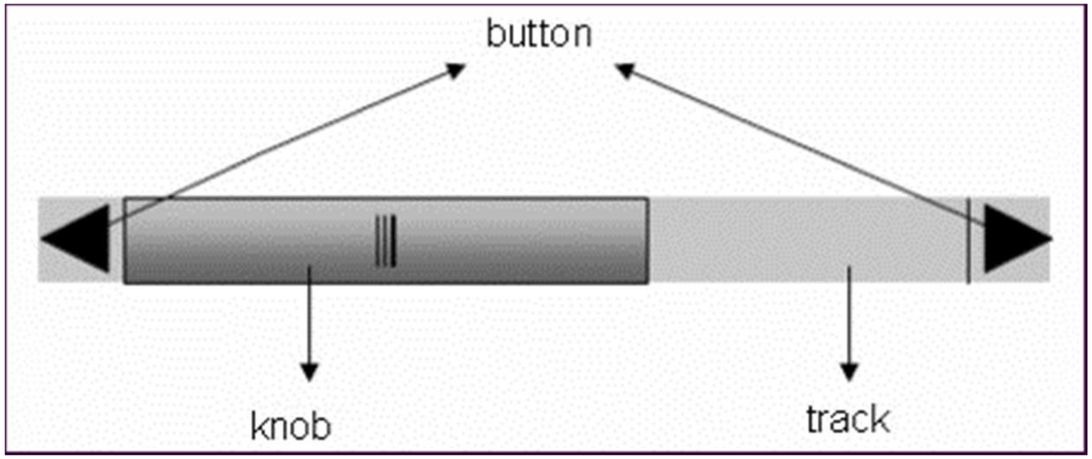
JScrollPane [3-4]

- The `JScrollPane` has a `JViewPort` which manages the visible area of the client.
- The viewport computes the bounds of the current visible area, based on the positions of the scroll bars.
- The scroll panes client is known as the view or viewport's view.
- The client of a scroll pane can be changed dynamically by invoking the `setViewportView()` method.
- Figure shows the viewport of `JScrollPane`.



JScrollPane [4-4]

- The three distinguished areas of a scroll bar are: knob, track, and button.



- On a vertical scroll bar as the user moves the knob, the visible area of the client moves up and down.
- Similarly, on a horizontal scroll bar as the user moves the knob, the visible area of the client moves back and forth.
- The knob position is proportionate to the visible area relative to the client.
- On clicking the arrow button, the user scrolls by unit increment whereas on clicking the track, the user scrolls by block increment.

Creating JScrollPane

- A `JScrollPane` is created by invoking its constructor and passing the component as an argument.
- The `JScrollPane` is then added to a container like any other components.
- Code Snippet shows how to add a `JTextArea` component to a `JScrollPane`.

Code Snippet

```
JFrame frmDisplay;
ScrollPane scpScrollPane;
JTextArea txaNotes;
. . .
   txaNotes = new JTextArea();
   frmDisplay = new JFrame("Scroll Pane");
// Provides the text area to the scroll pane
   scpScrollPane = new JScrollPane( txaNotes);
// Adds the scroll pane to the frame.
   frmDisplay.getContentPane().add( scpScrollPane);
```

Dynamically Changing the Clients Size

- The two steps to be followed by the programmer when the client of a `JScrollPane` is changed dynamically with the `setViewportView()` method are as follows:
    - Set the preferred size of the client with `setPreferredSize()` method
    - Invoke `revalidate()` method on the client
- The `revalidate()` method is required to be invoked so that the scroll pane can update itself and adjust the scroll bars.
- When the client size changes dynamically the scroll bars adjust automatically, however, the scroll pane or viewport does not resize.
- Code Snippet demonstrates how to dynamically change the client's size.

Code Snippet

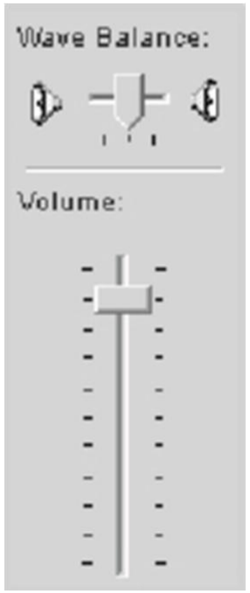
```
JPanel pnlClient;
JScrollPane scpScroller;
// Initialize the Client
pnlClient = new JPanel();
pnlClient.setSize(600,600);
// Add the components of the client
...
// Construct the scroll pane with the old view
   scpScroller = new JScrollPane(pnlClient);
...
// Resize the client
   pnlClient.setSize(600,800);
// Add more components to the client
...
   pnlClient.setPreferredSize(600,800);
   pnlClient.revalidate();
```

Methods of JScrollPane

- `public void setCorner(String key, Component corner)`: This method is used to set a component at the corner of the `JScrollPane`.
- `public void setHorizontalScrollBarPolicy(int policy)`: This method determines when the horizontal scroll bar appears in the scroll pane.
- `public void setVerticalScrollBarPolicy(int policy)`: This method determines when the vertical scroll bar appears in the scroll pane.
- `public void setRowHeaderView (Component view)`: This method is used to add a vertically long component as the row header. The component scrolls up and down when the vertical scroll bars start scrolling.
- `public void setColumnHeaderView (Component view)`: This method is used to add a horizontally long component as the column header. The component scrolls left and right when the horizontal scroll bars start scrolling.

JSlider

- A `JSlider` is a component which lets the user to select a numeric value within a bounded range, by sliding the slider on the slider bar.
- This component guarantees that the value selected by the user will always be in the specified range.
- Examples of slider bar usage are ‘Speaker Volume’ and ‘RGB Values for generating Color shade’.
- Advantage of using a slider bar for giving a numeric value is that you can preview the graphical effect of specifying the value dynamically.
- Figure displays a slider bar.
- Constructors:
  - `JSlider()`
  - `JSlider(int orientation)`
  - `JSlider(int min, int max)`
  - `JSlider(int min, int max, int value)`
  - `JSlider(int orientation, int min, int max, int value)`



Methods of JSlider [1-2]

- **`public int getValue()`**: Returns the current position of the slider in the slider bar as an integer.
- **`public void setValue(int value)`**: Sets the position of the slider bar programmatically. The value sent as the parameter should be within the range of minimum and maximum value of the slider bar.
- **`public void setOrientation(int orientation)`**: Is used to set the orientation of the slider bar to either horizontal or vertical.
- **`public void setMajorTickSpacing(int spacing)`**: Is used to set the major tick spacing. For example, if you have a slider with the range 0 to 20 and major tick spacing set to 10, then you will get ticks at 0, 10, and 20.
- **`public void setMinorTickSpacing(int spacing)`**: Is used to set the minor tick spacing. For example, consider a slider with the range 0 to 20, major tick spacing set to 10 and minor tick spacing set to 5. Then you will get 5 ticks between each major ticks.

Methods of JSlider [2-2]

- **`public void setSnapToTicks(boolean set)`**: Snaps the slider to the nearest tick mark wherever the user positions the slider.
- **`public void setPaintTicks(boolean set)`**: Determines whether the ticks are displayed on the slider bar. By default the ticks are not displayed.
- **`public void setPaintTrack(boolean set)`**: Determines whether the track of the slider bar should be painted.
- **`public void setPaintLabels(boolean set)`**: Determines whether the labels of the slider bar should be painted.

Event Handling of JSlider

- A `JSlider` component listens using the `javax.swing.event.ChangeListener` interface.
- A `JSlider` has the method `addChangeListener` to register a listener.
- This method is used to register a listener with the slider bar.
- The interface `ChangeListener` has a method `stateChanged`.
- Every time you move the slider, the slider bar fires an event and the control is delegated to this method.
- Code Snippet shows how to add the listener to the slider bar.

Code Snippet

```
JSlider sdrSilder;
. . .
sdrSilder.addChangeListener(new ChangeListener(){
       public void stateChanged(ChangeEvent ce){
       // Action code
       . . .
        . . .
        }
 });
```

Timer [1-2]

- A Swing Timer fires one or more ActionEvent events after a specified delay.
- A Timer can be configured to fire the event repeatedly or only once.
- To use a Timer, you have to specify the delay in milliseconds, and listener object of type ActionListener.
- **Use of Timer in Developing GUI**
  - Useful in GUI application especially where animations are used to trigger the displaying of the next frame in the animation.
  - Used in a GUI application where a user is supposed to respond in a stipulated time, for example, to answer a question.
- **Constructor Method**

```
Timer(int delay, ActionListener listener)
```

- **Start Timer**
  - To start the timer, you invoke the `start()` method of the `Timer` class.
  - Once started, the timer will fire the associated action events every specified time.

Timer [2-2]

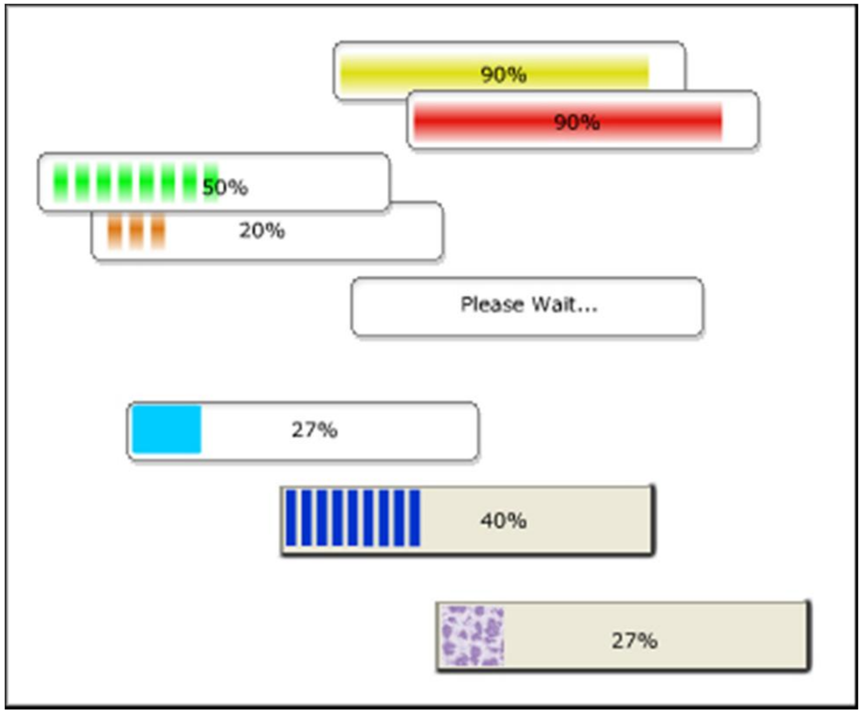
- Code Snippet shows how to create a timer which will repeatedly fire an action event every five seconds to display an alert message.

Code Snippet

```
Timer tmrAlert;
// Creates the timer with action event to be fired
//after every 5 seconds
tmrAlert = new Timer(5000, new ActionListener() {
        public void actionPerformed(ActionEvent ae)
{
                // Action Code
                . . .
                // Display the Alert Message
                . . .
         }
 });
```

JProgressBar [1-3]

- A progress bar typically indicates the progress of a time consuming event by displaying its percentage of completion.
- Normally progress bars are used in splash screens to display the loading status of an application.
- They are also used to display the progress of a time consuming operation such as copying files from one location to another.
- Figure displays the progress bar.



JProgressBar [2-3]

- Swing component `JProgressBar` displays the progress of any task.
- For example, it can display a progress bar indicating the installation progress in percentage terms.
- **Constructors:**
  - `JProgressBar()`
  - `JProgressBar(int orient)`
  - `JProgressBar(int min, int max)`
  - `JProgressBar(int orient, int min, int max)`
- Code Snippet demonstrates how to use an indeterminate type progress bar.

Code Snippet

```
JProgressBar pgb = new JProgressBar();
// to set indeterminate type progress bar assign
//boolean value to true otherwise false
  pgb.setIndeterminate(true);
```

JProgressBar [3-3]

- Code Snippet shows how to create a vertically oriented progress bar with 0 as minimum value, 100 as maximum value, and add it to the frame.

Code Snippet

```
. . .
JProgressBarDemo() {
// Creates a panel container
    JPanel jPanel = new JPanel();
// Creates progress bar with minimum value 0and maximum value 100
    JProgressBar pgbLongTask = new JProgressBar(JProgressBar.VERTICAL,
0, 100);
    pgbLongTask.setValue(50);
    jPanel.add(pgbLongTask);
// Adds panel to the frame
    add(jPanel);
// Sets the attributes of the frame
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    pack();
. . .
```

Updating the Progress Bar [1-2]

- Once the progress bar is created and displayed, you invoke the `setValue()` method of the progress bar to update the progress.
- The argument sent in the method should be within the bounded range of the progress bar.
- Typically, a Timer is used to determine the amount of task completed and accordingly update the progress bar.

Code Snippet

```
JProgressBar pgbLongTask;
JFrame frmProgress;
Timer tmrProgress;
int taskCompleted = 0;

// Creates the progress bar with the minimum value 0 and maximum
// value 100
   pgbLongTask = new JProgressBar(0, 100);

// Adds the progress bar to the frame
   frmProgress.getContentPane().add(pgbLongTask);
```

Updating the Progress Bar [2-2]

```
// Creates the timer with the action event to be fired after every 2 //seconds
  tmrProgress = new Timer(2000, new ActionListener() {
  public void actionPerformed(ActionEvent ae)
  {
   // Action Code
  // Compute the percentage of task completed
     taskCompleted = 50;
  // Update the progress bar with the percentage of
  // task completed
     taskCompleted = 50;
   // Update the progress bar with the percentage of
   // task completed
      pgbLongTask.setValue(taskCompleted);
                  . . .
  }
 });
```

Methods of JProgressBar [1-3]

- **`void setMinimum(int n)`**: This method is used to set the minimum value of the progress bar.
- **`void setMaximum(int n)`**: This method is used to set the maximum value of the progress bar.
- **`void setValue(int n)`**: This method is used to set the progress value of the progress bar.
- **`void setString(String s)`**: This method is used to set the string representation of the numeric value of progress. Once set the progress bar will use this value followed by the percentage sign. The `setStringPainted()` method decides whether the string will be displayed or not.
- **`void setStringPainted(boolean b)`**: This method determines whether the string representation of the numeric value is to be displayed or not.

Methods of JProgressBar [2-3]

- Table lists other methods of `JProgressBar`.

Method	Description
`void setIndeterminate(boolean)`	Setting boolean value to 'True' will make indeterminate progress bar. Setting Boolean value to 'false' will make it determinate, which is default.
`void setOrientation(int)`	Sets a progress bar horizontal or vertical by specifying JProgressBar.HORIZONTAL or JProgressBar.VERTICAL.
`int getOrientation()`	Gets a horizontal/vertical progress bar
`String getString()`	Gets percentage string
`public int getValue()`	Returns current value from BoundedRangeModel.
`public int getMinimum()`	Returns current minimum value from BoundedRangeModel.
`public int getMaximum()`	Returns current maximum value from BoundedRangeModel.
`protected ChangeListener createChangeListener()`	Used to implement custom ChangeListener.
`public void addChangeListener (ChangeListener cI)`	Adds specified ChangeListener to progress bar.

Methods of JProgressBar [3-3]

Method	Description
`public void removeChangeListener(ChangeListener cI)`	Removes ChangeListener from progress bar.
`public BoundedRangeModel getModel()`	Returns model used by the progress bar.
`public void setModel(Bounded RangeModel newModel)`	Replaces current model used by progress bar.
`protected void fireStateChanged()`	Fires events of type ChangeEvent to its registered listeners. Only subclasses of JProgressBar can invoke it.

Event Handling of JProgressBar

- The `JProgressBar` component listens using the `ChangeListener` interface.
- A `JProgressBar` has the `addChangeListener` method to register a listener.
- The interface `ChangeListener` has a method `stateChanged()`.
- Every time the progress value of progress bar is updated, it fires an event and the control is delegated to this method.
- To retrieve the current value of the progress bar, you use the `getValue()` method.
- Code Snippet shows how to add a listener to the progress bar.

Code Snippet

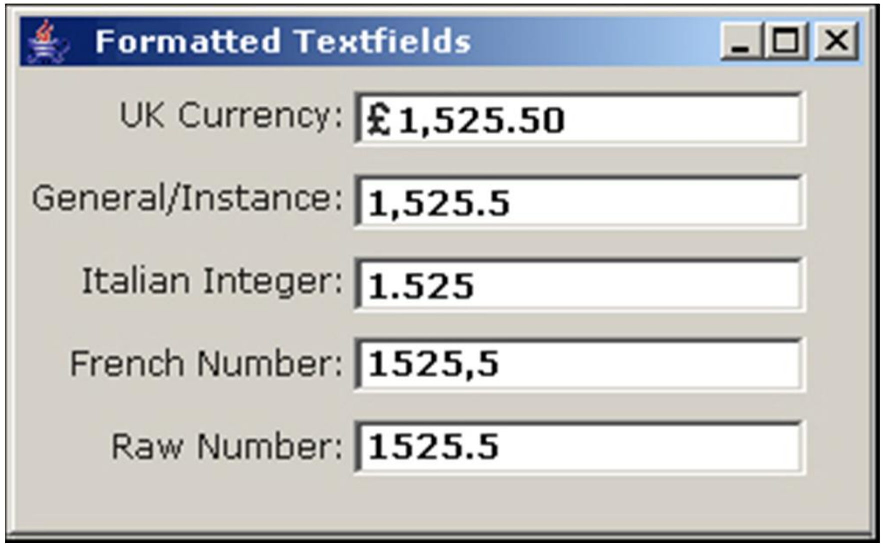
```
pgbLongTask.addChangeListener(new ChangeListener()
{
        public void stateChanged(ChangeEvent ce) {
                // Action Code
                . . .
                . . .
        }
});
```

Advanced Text Components

- Advanced text components are used for text manipulations.
- These components inherit from `JTextComponent` class and provide other services for additional flexibility.
- Some of the advanced components are as follows:
  - **`JFormattedTextField`** - Extends `JTextField`, adding support for formatting arbitrary values, as well as retrieving a particular object once the user has edited the text.
  - **`JEditorPane`** - Is a text component to edit various kinds of content. This component uses implementations of the Editor Kit to accomplish its behavior.
  - **`JTextPane`** - Is a component that can be marked up with attributes that are represented graphically.

JFormattedTextField

- The `JFormattedTextField` is similar to a `JTextField` in appearance.
- It accepts characters based on the formatter used.
- Different formatters are available such as numeric, date, and currency.
- In case you use a `JTextField` component in a GUI application, then you are responsible to check the value that is typed in the text field for proper format.
- `JFormattedTextField` makes it convenient to do the validation of an input field without writing any code to check the value entered in it.
- Figure displays the `JFormattedTextField`.



UK Currency: £1,525.50
General/Instance: 1,525.5
Italian Integer: 1.525
French Number: 1525,5
Raw Number: 1525.5

Using JFormattedTextField with Numeric Formatter [1-2]

To use a `JFormattedTextField` with a numeric formatter, you will follow these steps:

1
- Declare an object of `NumberFormat` class.

2
- Declare an object of `JFormattedTextField` class.

3
- Use the static method `NumberFormat.getNumberInstance()` to create an instance of `NumberFormat`.

4
- Create an instance of `JFormattedTextField` and pass the instance of `NumberFormat` as an argument to it.

5
- Add the instance of `JformattedTextField` to the container.

Using JFormattedTextField with Numeric Formatter [2-2]

- Code Snippet shows how to create the `JFormattedTextField` with numeric formatter and add it to the frame.

Code Snippet

```
NumberFormat numberFormat;
JFormattedTextField txfNumber;
JFrame frmDetails;
// Instantiate a number format
   object numberFormat =
NumberFormat.getNumberInstance();
// Creates the formatted text field with numeric
formatter
   txfNumber = new JFormattedTextField(numberFormat);
// Adds the formatted text field to the frame
   frmDetails.getContentPane().add(txfNumber);
```

Using JFormattedTextField with Date Formatter

- To use a `JFormattedTextField` with a date formatter, you will follow these steps:
  - Declare an object of `DateFormat` class.
  - Declare an object of `JFormattedTextField` class.
  - Use the static method `DateFormat.getDateInstance()` to create an instance of `DateFormat` class.
  - Create an instance of `JFormattedTextField` and pass the instance of `DateFormat` as an argument to it.
  - Add the instance of `JFormattedTextField` to the container.
  - Code Snippet shows how to create the `JFormattedTextField` with date formatter.

Code Snippet

```
DateFormat dateFormat;
JFormattedTextField txfDate;
JFrame frmDetails;
// Instantiates a dateFormat object
   dateFormat = DateFormat.getDateInstance();
// Creates the formatted text field with date formatter
    txfDate = new JFormattedTextField(dateFormat);
// Adds the formatted text field to the frame
    frmDetails.getContentPane().add(txfDate);
```

Using JFormattedTextField with Customized Formatter

- To create a formatted text field with a customized format, you use the `MaskFormatter` class.
- Table lists the mask characters.

Mask Character	Description
`#`	Any valid number.
`` ` ``	Escape character for escape sequence.
`U`	Any character, converted to uppercase.
`L`	Any character, converted to lowercase.
`A`	Any character or number.
`?`	Any single character.
`*`	One or more characters.
`H`	Any hex character(0-9, a-f, or A-F).

- Example: `formatter = new MaskFormatter("###,###,###");`

JEditorPane [1-2]

- A `JEditorPane` is a text component which can display and edit text of type plain, HTML, and RTF.
- The text can be of various styles intermixed throughout the editor pane.
- Constructors:
  - `JEditorPane()`
  - `JEditorPane(String url) throws IOException`
  - `JEditorPane(URL url) throws IOException`
  - `JEditorPane(String type, String text)`
- An HTML text is displayed in a non-editable `JEditorPane`. If a `JEditorPane` is non-editable then it supports hyperlink events.
- A `JEditorPane` can be set to be non-editable by invoking the `setEditable()` method and passing boolean value false.
- To handle the hyperlink events, Swing provides the `javax.swing.event.HyperlinkListener` interface.

JEditorPane [2-2]

- The methods of `JEditorPane` are:
  - **`public void setText(String text)`** :
    - This method is used to set the text in the JEditorPane.

Code Snippet

```
String info = "Once upon a time,……";
 // Sets the text in the editor pane
epnEditor.setText(info);
```

  - **`public void setPage(URL url) throws IOException`**:
    - This method is used to display the contents of the page referenced by the url.

Code Snippet

```
// Creates an URL object with the specified url.
URL url = new URL("http://java.sun.com/index.html");
// Displays the contents of the URL in the JEditorPane.
epnEditor.setPage(url);
```

- To display the `JEditorPane`, pass an instance of `JEditorPane` to the constructor of a `JScrollPane`, and add the `JScrollPane` to the container.

Example of JEditorPane [1-4]

- Code Snippet demonstrates the use of `JEditorPane`.

Code Snippet

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.html.HTMLFrameHyperlinkEvent;
import javax.swing.text.html.HTMLDocument;

public class ReadURLFileNew extends JFrame {
         private JTextField urlValue;
         private JEditorPane contents;
         public ReadURLFileNew() {
                  super( "Web Browser" );

                  Container conObj = getContentPane();
                  urlValue = new JTextField( "Enter URL here" );
                  urlValue.addActionListener(
                            new ActionListener() {
```

Example of JEditorPane [2-4]

```
   public void actionPerformed( ActionEvent e ) {
                     getPages( e.getActionCommand() );
  }
 }  );
          conObj.add( urlValue, BorderLayout.NORTH );
          contents = new JEditorPane();
          contents.setEditable( false );
          contents.addHyperlinkListener(
          new HyperlinkListener() {
                     public void hyperlinkUpdate(HyperlinkEvent e) {
                     if (e.getEventType()== HyperlinkEvent.EventType.Activated)
                      getPages(e.getURL().toString());
                     if (e instanceof HTMLFrameHyperlinkEvent) {
                                  HTMLFrameHyperlinkEvent  evt =
          (HTMLFrameHyperlinkEvent)e;
                     HTMLDocument doc = (HTMLDocument)contents.getDocument();
                     doc.processHTMLFrameHyperlinkEvent(evt);
                     } else {
                              try {
                              contents.setPage(e.getURL());
```

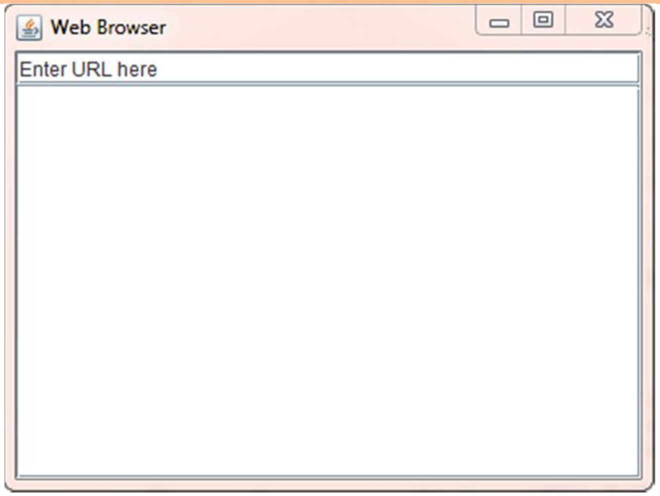
Example of JEditorPane [3-4]

```
                              } catch (Exception ex) {
                               ex.printStackTrace();
                               }
                      }
                   }
                }     );
                conObj.add( new JScrollPane( contents ),
                BorderLayout.CENTER );
                setSize( 400, 300 );
                setVisible(true);
}
 private void getPages( String location ) {
                 setCursor( Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR ) );
                 try {
                contents.setPage( location );
                urlValue.setText( location );
                 } catch ( IOException io ) {
                JOptionPane.showMessageDialog( this,  “Error retrieving data from the
                 specified URL site”,  “Check URL”, JOptionPane.ERROR_MESSAGE );
```

Example of JEditorPane [4-4]

```
                    }
                    setCursor( Cursor.getPredefinedCursor(
Cursor.DEFAULT_CURSOR ) );
}
public static void main( String args[] ) {
          ReadURLFileNew urlObj = new ReadURLFileNew();
          urlObj.addWindowListener(
          new WindowAdapter() {
                    public void windowClosing( WindowEvent e )
                    {
                              System.exit( 0 );
                    }
           });
}
}
```

- Output:



JTextPane [1-2]

- A `JTextPane` is a text component which supports styled text.
- It is similar to a `JTextArea` in appearance.
- The same font is applied throughout the text of a text area.
- A `JTextPane` can have plain-text, HTML, RTF (Rich Text Format) and even embedded components such as buttons and icons.
- **Constructors**:
  - `JTextPane()`
  - `JTextPane(StyledDocument doc)`
- The steps to load the text in the `JTextPane` are as follows:

1. Retrieve the `StyledDocument` of a `JTextPane` is retrieved using the method `getStyledDocument()`.
2. Create and initialize styles.
3. Add styles to the style-document.
4. Load textual data - To load textual data the `insertString()` method of styled-document is used.

JTextPane [2-2]

- To display the `JTextPane`, pass an instance of `JTextPane` to the constructor of a `JScrollPane`.
- Then add the `JScrollPane` to the container. To disable editing of a JTextPane, use the `setEditable()` method. To disable editing set 'false' as an argument.

Code Snippet

```
JFrame frmEditor;
JTextPane txpPane;
…
// Provide the text pane to the scroll pane to
facilitate scrolling
JScrollPane scpScroller = new JScrollPane(txpPane);
// Adds the JScrollPane to the frame
frmEditor.getContentPane().add(scpScroller);
//Disables the editing of the JTextPane
txpPane.setEditable(false);
```

Applying Icons and Borders [1-2]

- The Swing components, such as labels and buttons can be associated with icons and borders.
- Icon is a picture or image that can be applied to a component. Image can be of different formats, such as GIF, JPEG, or PNG.
- Similarly, borders are used to apply fancy edges to the Swing components.
- **`ImageIcon API`**
  - Provides a utility class named `ImageIcon` which allows you to specify an image for the components.
  - The image to be inserted as icon and then associated with the component.
  - Thus, to apply an icon from existing images to a Swing component, you need to create an object of type `ImageIcon` class.
- **`Constructors`**
  - `ImageIcon()`
  - `ImageIcon(Byte[] image data)`
  - `ImageIcon(Image image)`
  - `ImageIcon(Image image, String description)`
  - `ImageIcon(string filename)`
  - `ImageIcon(URL location)`

Applying Icons and Borders [2-2]

- Code Snippet shows how to apply an image to the label component using `ImageIcon` methods.

Code Snippet

```
ImageIcon i = createImageIcon(“pic.gif”, “this is my
image”);
Label a = new JLabel(“Image and Text”, i, JLabel.CENTER);
...
Label2 = new JLabel(i)
```

- Code Snippet shows how to create an image icon from the specified resource, such as a URL.

Code Snippet

```
java.net.URL imgURL =
myFile.class.getResource(“mypic.gif”);
...
if (imgURL != null) {
ImageIcon i = new ImageIcon(imgURL);
```

Loading Images in Applet [1-2]

- The `JApplet` supports the swing graphic library.
- An applet loads image data from the same system that serves up the applet.
- The `JApplet` can load images that are in the GIF or JPG format. The images can be loaded using `init()` method.

  Syntax:

  ```
  public void init();
  ```

- Code Snippet shows how to load and display an image in `JApplet`.

Code Snippet

```
public class animal extends Applet
{
private Image lion;
public void init()
{
    lion = null;
}
```

Loading Images in Applet [2-2]

```
public void loadImage()
{
  try
  {
   lion = getImage(getDocumentBase(), "image1.gif");
  }  catch(Exception e) { . . . }
}
public void paint(Graphics g)
 {
    if (img == null)
    loadImage();
    g.drawImage(img, 0, 0, this);
 }
}
```

Border API [1-2]

- The Border API can be used to apply the borders to the components. There are two ways to create the border that are as follows:
  - Create a new border using the `BorderFactory` class.
  - Using setter or getter method to apply borders.
- **Creating a Border Using BorderFactory**
  - The `BorderFactory` is a class that returns references for the `Border` objects.
  - The `BorderFactory` class is used to create borders for components.
  - The `BorderFactory` class has static methods defines in it to create predefined borders.
  - Some of the methods to create borders using the `BorderFactory` class:
    - `void setBorder(Border)`
    - `BordercreateLineBorder(color)`
    - `createLineBorder(color, int)`
    - `BordercreateEtchedBorder()`
    - `BordercreateemptyBorder(int, int, int, int)`
    - `public static TitledBorder createTitledBorder(String title)`

Border API [2-2]

- Code Snippet shows how to create and set simple borders using the `BorderFactory` class.

Code Snippet

```
// Sets simple borders
     blkln = BorderFactory.createLineBorder(Color.black);
    retched =
    BorderFactory.createEtchedBorder(EtchedBorder.RAISED);
    lowbev = BorderFactory.createLoweredBevelBorder();
    emp = BorderFactory.createEmptyBorder();
    jCompa.setBorder(blkln);
    jCompb.setBorder(retched);
    jcompc.setBorder(lowbev);
    jCompd.setBorder(emp);
```

Dialog Box

- A Swing based application is created with `JFrame` as the top-level container.
- This container is physically constrained with the maximum size offered by the screen on which it is displayed.
- Complex and large applications cannot display all its GUI components on a single frame.
- Several intermediate frames are required to pop up based on different events in the application.
- These pop-up frames are called Dialog boxes.
- The parent of a dialog box is typically the `JFrame`.
- Dialog boxes are of two types:
  - **Modal**: Modal dialog box block their parent when they pop up. You have to finish the work and close the dialog box to return back to the parent.
  - **Non-Modal**: Non-modal dialog box do not block their parent. If you click their parent, the dialog box disappears.

JOptionPane

- The Swing component `JOptionPane` is a representation of option panes.
- A option pane is a type of dialog box that allows the user to enter options, and depending on options selected a program can be executed further.
- This component can be typically used to display feedback message or confirmation or to input information from users.
- `JOptionPane` is the subclass of JComponent class.
- The different types of messages supported by `JOptionPane` are:

1
- ERROR_MESSAGE – Used for error message.

2
- INFORMATION_MESSAGE – Used for information message.

3
- WARNING_MESSAGE – Used for warning message.

4
- QUESTION_MESSAGE – Used for questions.

5
- PLAIN_MESSAGE – No icon is used.

Different Types of Dialog Boxes Using JOptionPane [1-2]

- **InputDialog**
  - This dialog is used to present the user with a drop-down having multiple choices to select from or a text field to type a value.
  - The `JOptionPane` has the static method `showInputDialog()` to show this dialog.

Code Snippet

```
JOptionPane.showMessageDialog(null, “Your License has
expired.”,  “Security  warning”,
JOptionPane.WARNING_MESSAGE);
```

- **ConfirmDialog**
  - You use this dialog box to ask the user a confirming question with a ‘Yes’, ‘No’, or ‘Cancel’ button.
  - You use this dialog box to get a confirmation from the user.
  - The `JOptionPane` has the static method `showConfirmDialog()` to show this dialog.

Code Snippet

```
int n = JOptionPane.showConfirmDialog(null,”Would you like
to continue?”, “Confirmation”, JOptionPane.YES_NO_OPTION);
```

Different Types of Dialog Boxes Using JOptionPane [2-2]

- **MessageDialog:**
  - This dialog box is used to alert the user with some message.
  - The `JOptionPane` has the static method `showMessageDialog()` to display this dialog.

Code Snippet

```
JOptionPane.showMessageDialog(null, "Your License has
expired.", "Security  warning",
JOptionPane.WARNING_MESSAGE);
```

- **OptionDialog:**
  - This dialog allows you to change the text that appears on the buttons of standard dialogs.
  - The `JOptionPane` has the static method `showOptionDialog()` to show this dialog.

Code Snippet

```
 Object[] options = {"Connect", "Disconnect","Quit"}; i
nt n = JOptionPane.showOptionDialog(null,"What would you like to  do?",
 "Connection Message", JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,  null,  options, options[0]);
```

JDialog [1-2]

- A `JDialog` is a top-level container to create custom dialog boxes.
- The appearance is almost similar to a frame except that it does not have a minimize button.
- The process of creating, displaying, and closing of a `JDialog` is almost identical to a `JFrame`.
- **Constructors:**
  - `JDialog(Frame parent, String title)`
  - `JDialog(Frame parent, String title, boolean modal)`
- Components are added to the content-pane of a `JDialog`.
- By default, a `JDialog` box is governed by a `BorderLayout` manager.
- To display the dialog box, you invoke the `setVisible()` method.
- After this method invocation, you should not add any components to the dialog box.
- To close the dialog box, you can invoke the `setDefaultCloseOperation()` method with the parameter `JDialog.DISPOSE_ON_CLOSE`.

JDialog [2-2]

- Code Snippet shows how to create a search dialog.

Code Snippet

```
public class SearchDialog extends JDialog {
         JButton btnSearch;
         public SearchDialog()       {
           btnSearch = new JButton("Search");
            getContentPane.add(btnSearch);
           // Closes the dialog box
               setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
            // Handle events
                   . . .
                   . . .
         }
}
```

Summary

- JScrollPane provides a scrollable view of a component.
- A JSlider is a component, which lets the user to select a numeric value within a bounded range, by sliding a slider on the component.
- JProgressBar indicates the progress of a time consuming event by displaying its percentage of completion.
- A JFormattedTextField allows only legal characters to be input in a proper format.
- A JEditorPane is a text component which can display and edit text of type plain, HTML and RTF. A JTextPane is a text component which supports styled text.
- The ImageIcon class implements the Icon interface and allows applying images to the components.
- JOptionPane provides a convenient means to display a standard dialog box for user inputs and alert messages.