Applets and Swing

Concepts of Applets and Differences from Applications

1. **Concepts of Applets**:
   - **Definition**: Applets are small Java programs that are designed to run within a web browser or applet viewer. They are used to provide interactive features on web pages.
   - **Execution**: Applets are executed by an applet viewer or a web browser with Java support.
   - **Security**: Applets run in a restricted environment (sandbox) for security reasons, limiting their access to system resources.

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

public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello, Applet!", 20, 20);
    }
}
```

2. **Differences from Applications**:
   - **Execution Environment**:
     - **Applets**: Run in a browser or applet viewer. They are part of a web page and are controlled by the browser's security policy.
     - **Applications**: Run independently and directly on the Java Virtual Machine (JVM). They have more control over system resources.
   - **Entry Point**:
     - **Applets**: Do not have a `main` method. They have a predefined life cycle with methods like `init()`, `start()`, `stop()`, and `destroy()`.
     - **Applications**: Have a `main` method which is the entry point of the application.
   - **User Interface**:
     - **Applets**: Typically used to create small, interactive features on web pages.
     - **Applications**: Used for standalone applications with complex user interfaces.
   - **Security**:
     - **Applets**: Run in a sandbox environment with restricted access to system resources for security.

- **Applications**: Have more access to system resources and are not restricted in the same way as applets.

Life Cycle of an Applet

The life cycle of an applet involves the following methods, which are called by the browser or applet viewer at various stages:

1. `init()`:
   - **Purpose**: Called once when the applet is first loaded into memory. Used for initialization, such as setting up user interface components.
   - **Example**:

```
public void init() {
    // Initialization code
}
```

2. `start()`:
   - **Purpose**: Called after `init()` to start or resume the applet. Used to start animations or other activities.
   - **Example**:

```
public void start() {
    // Code to start the applet
}
```

3. `paint(Graphics g)`:
   - **Purpose**: Called whenever the applet needs to be redrawn, such as when it is first loaded or when it is resized. Used for drawing on the applet's display area.
   - **Example**:

```
public void paint(Graphics g) {
    g.drawString("Hello, Applet!", 20, 20);
}
```

4. `stop()`:
   - **Purpose**: Called when the applet is no longer visible or needs to be suspended. Used to stop any ongoing activities like animations.
   - **Example**:

```
public void stop() {
    // Code to stop activities
}
```

5. `destroy()`:
   - **Purpose**: Called when the applet is being unloaded from memory. Used for cleanup, such as releasing resources.
   - **Example**:

```
public void destroy() {
    // Cleanup code
}
```

Types of Applets

1. `Applet`:
   - **Definition**: The base class for all applets. It provides the basic functionality needed for applets, including life cycle methods and drawing capabilities.
2. `AudioClip` **Applet**:
   - **Definition**: Applets that play audio. They use the `AudioClip` interface to handle sound playback.
   - **Example**:

```
import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;

public class AudioApplet extends Applet {
    private AudioClip sound;

    public void init() {
        URL url = getCodeBase();
        sound = getAudioClip(url, "sound.wav");
    }

    public void start() {
        sound.loop(); // Play sound in a loop
    }

    public void stop() {
        sound.stop(); // Stop sound playback
```

```
    }
}
```

3. `Applet` **with GUI Components**:
   - **Definition**: Applets that include GUI components like buttons, text fields, and panels.
   - **Example**:

```
import java.applet.Applet;
import java.awt.Button;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GUIApplet extends Applet {
    public void init() {
        Button button = new Button("Click Me");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button Clicked");
            }
        });
        add(button);
    }
}
```

4. `Interactive` **Applet**:
   - **Definition**: Applets that interact with the user through input events like mouse clicks or keyboard input.
   - **Example**:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class InteractiveApplet extends Applet implements MouseListener
{
    public void init() {
        addMouseListener(this);
    }

    public void mouseClicked(MouseEvent e) {
        repaint(); // Trigger repaint on mouse click
    }

    public void paint(Graphics g) {
```

```
        g.drawString("Click anywhere", 20, 20);
    }

    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}
```

Creating Applets and Passing Parameters

1. **Creating Applets**:
   - To create an applet, extend the `Applet` class (or `JApplet` for Swing-based applets) and override the necessary lifecycle methods (`init()` , `start()` , `stop()` , `destroy()`).
   - **Example**:

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

public class SimpleApplet extends Applet {
    public void init() {
        // Initialization code
    }

    public void paint(Graphics g) {
        g.drawString("Hello, Applet!", 20, 20);
    }
}
```

   - **HTML for Embedding an Applet**:

```
<html>
<body>
    <applet code="SimpleApplet.class" width="300" height="200">
    </applet>
</body>
</html>
```

2. **Passing Parameters to Applets**:
   - Applets can receive parameters from the HTML file in which they are embedded. These parameters can be retrieved using the `getParameter()` method.

- **Example**:

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

public class ParameterApplet extends Applet {
    private String message;

    public void init() {
        message = getParameter("message");
        if (message == null) {
            message = "Default Message";
        }
    }

    public void paint(Graphics g) {
        g.drawString(message, 20, 20);
    }
}
```

- **HTML with Parameters**:

```
<html>
<body>
    <applet code="ParameterApplet.class" width="300" height="200">
        <param name="message" value="Hello from Parameter!">
    </applet>
</body>
</html>
```

Introduction to Swing and Limitations of AWT

1. **Introduction to Swing**:
   - **Swing**: A part of Java's standard library, Swing provides a more sophisticated set of GUI components than AWT. It is built on top of AWT and provides a more flexible and powerful toolkit for creating modern user interfaces.
   - **Key Features**:
     - **Pluggable Look-and-Feel**: Allows applications to have a consistent appearance across different platforms.
     - **Lightweight Components**: Swing components are lightweight, meaning they are not dependent on native system resources for rendering.
     - **MVC Architecture**: Swing uses the Model-View-Controller (MVC) design pattern, which separates data (model) from user interface (view) and controls

(controller).

- **Basic Swing Example**:

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.FlowLayout;

public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());

        JButton button = new JButton("Click Me");
        frame.add(button);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

2. **Limitations of AWT**:

- **Heavyweight Components**: AWT components are heavyweight, meaning they rely on native system resources for rendering. This can result in inconsistent look and feel across different platforms.
- **Limited Customization**: AWT provides limited options for customizing the appearance of components compared to Swing.
- **Look-and-Feel**: AWT does not support pluggable look-and-feel, meaning that components appear with the native OS appearance.
- **Event Handling**: AWT's event handling model is less flexible and more cumbersome compared to Swing's event handling.
- **Deprecated Components**: Some AWT components are considered outdated and have been replaced by newer Swing components.

Comparison Example:

- **AWT Frame Example**:

```
import java.awt.Frame;
import java.awt.Button;

public class AWTExample {
    public static void main(String[] args) {
        Frame frame = new Frame("AWT Example");
        frame.setLayout(new FlowLayout());
```

```
        Button button = new Button("Click Me");
        frame.add(button);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

- **Swing Frame Example**:

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.FlowLayout;

public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());

        JButton button = new JButton("Click Me");
        frame.add(button);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

- **Swing Benefits**:
  - Better appearance and consistency across platforms.
  - Greater flexibility and customization options.
  - Richer set of GUI components.

MVC Architecture and Components in Swing

MVC Architecture

Model-View-Controller (MVC) is a design pattern used to separate the concerns of an application, making it more modular and easier to manage. In Swing, MVC helps in organizing the application into three distinct components:

1. **Model**:

- **Purpose**: Represents the data and business logic of the application. It manages the data and notifies the view about any changes in the data.
- **Example**: A class that manages the data of a data table or a class that handles the application's business logic.
- **In Swing**: Swing components like `JTable` and `JList` use models to manage and represent their data.

2. **View**:
   - **Purpose**: Represents the graphical user interface (GUI) of the application. It displays the data and sends user commands to the controller.
   - **Example**: GUI components like buttons, labels, and text fields.
   - **In Swing**: Swing provides a wide range of view components like `JButton`, `JLabel`, `JTextField`, etc., to display and interact with the data.
3. **Controller**:
   - **Purpose**: Acts as an intermediary between the model and the view. It receives user input from the view, processes it, and updates the model and/or the view accordingly.
   - **Example**: Event listeners like `ActionListener` and `MouseListener` that handle user interactions.
   - **In Swing**: Controllers are often implemented using event listeners. For example, an `ActionListener` for a button will handle the button click event and update the model or view as needed.

Components in Swing

Swing provides a rich set of components for building modern graphical user interfaces. Here are some key Swing components:

1. **JFrame**:
   - **Purpose**: Represents a top-level window with standard decorations (title bar, border, etc.).
   - **Example**:

```
import javax.swing.JFrame;

public class MyFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Frame");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

2. **JPanel**:
   - **Purpose**: A container used to group and organize other components.
   - **Example**:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

public class PanelExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Panel Example");
        JPanel panel = new JPanel();
        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        frame.add(panel);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

3. **JButton**:
   - **Purpose**: Represents a push button that can trigger an action when clicked.
   - **Example**:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Button Example");
        JButton button = new JButton("Click Me");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button Clicked");
            }
        });
        frame.add(button);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

4. **JLabel**:
    - **Purpose**: Displays a non-editable text or image.
    - **Example**:

```
import javax.swing.JLabel;
import javax.swing.JFrame;

public class LabelExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Label Example");
        JLabel label = new JLabel("Hello, Swing!");
        frame.add(label);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

5. **JTextField**:
    - **Purpose**: Allows the user to enter a single line of text.
    - **Example**:

```
import javax.swing.JTextField;
import javax.swing.JFrame;

public class TextFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("TextField Example");
        JTextField textField = new JTextField("Enter text here");
        frame.add(textField);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

6. **JTextArea**:
    - **Purpose**: Allows the user to enter multiple lines of text.
    - **Example**:

```
import javax.swing.JTextArea;
import javax.swing.JFrame;

public class TextAreaExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("TextArea Example");
```

```
        JTextArea textArea = new JTextArea("Enter multiple lines of
text");
        frame.add(textArea);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

7. **JCheckBox**:
   - **Purpose**: Allows the user to select or deselect an option.
   - **Example**:

```
import javax.swing.JCheckBox;
import javax.swing.JFrame;

public class CheckBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("CheckBox Example");
        JCheckBox checkBox = new JCheckBox("Check Me");
        frame.add(checkBox);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

8. **JRadioButton**:
   - **Purpose**: Represents a radio button that allows the user to select one option from a group.
   - **Example**:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;

public class RadioButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("RadioButton Example");
        JPanel panel = new JPanel();
        JRadioButton option1 = new JRadioButton("Option 1");
        JRadioButton option2 = new JRadioButton("Option 2");
        ButtonGroup group = new ButtonGroup();
        group.add(option1);
        group.add(option2);
```

```
        panel.add(option1);
        panel.add(option2);
        frame.add(panel);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

9. **JList**:
    - **Purpose**: Displays a list of items that can be selected.
    - **Example**:

```
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;

public class ListExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("List Example");
        String[] data = {"Item 1", "Item 2", "Item 3"};
        JList<String> list = new JList<>(data);
        frame.add(new JScrollPane(list));
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

10. **JTable**:
    - **Purpose**: Displays tabular data.
    - **Example**:

```
import javax.swing.JFrame;
import javax.swing.JTable;
import javax.swing.JScrollPane;

public class TableExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Table Example");
        String[] columns = {"Name", "Age"};
        Object[][] data = {
            {"Alice", 30},
            {"Bob", 25}
        };
        JTable table = new JTable(data, columns);
        frame.add(new JScrollPane(table));
```

```
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Swing components are more advanced and provide a greater degree of customization compared to AWT components. They are also designed to be more portable and consistent across different platforms.

Exploring Swing Components

Swing offers a rich set of components for building graphical user interfaces in Java. Here's an overview of some key Swing components and how to use them:

JApplet

- **Purpose**: Provides the applet functionality for Swing-based applets.
- **Usage**: Although applets are largely obsolete, `JApplet` was used for embedding Swing components into web pages.
- **Example**:

```
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JPanel;

public class SwingApplet extends JApplet {
    public void init() {
        JPanel panel = new JPanel();
        JButton button = new JButton("Click Me");
        panel.add(button);
        add(panel);
    }
}
```

JFrame

- **Purpose**: Represents a top-level window with standard decorations (title bar, border, etc.).
- **Usage**: Used as the main window of a Swing application.

- **Example**:

```
import javax.swing.JFrame;
import javax.swing.JButton;

public class JFrameExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JFrame Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);
        JButton button = new JButton("Click Me");
        frame.add(button);
        frame.setVisible(true);
    }
}
```

JComponent

- **Purpose**: Base class for all Swing components that have a graphical representation. It provides basic functionality and properties for all Swing components.
- **Usage**: You typically don't use `JComponent` directly but extend it to create custom components.
- **Example**:

```
import javax.swing.JComponent;
import java.awt.Graphics;

public class CustomComponent extends JComponent {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Custom Component", 20, 20);
    }
}
```

Icons

- **Purpose**: Used to represent images or graphics in Swing components.
- **Usage**: Commonly used with buttons and labels to display images.
- **Example**:

```
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;

public class IconExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Icon Example");
        ImageIcon icon = new ImageIcon("path/to/icon.png");
        JButton button = new JButton(icon);
        frame.add(button);
        frame.setSize(200, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JLabel

- **Purpose**: Displays text or an image.
- **Usage**: Used for displaying static text or images.
- **Example**:

```
import javax.swing.JLabel;
import javax.swing.JFrame;

public class JLabelExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JLabel Example");
        JLabel label = new JLabel("Hello, JLabel!");
        frame.add(label);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JTextField

- **Purpose**: Allows the user to input a single line of text.
- **Usage**: Used for accepting user input.
- **Example**:

```
import javax.swing.JTextField;
import javax.swing.JFrame;

public class JTextFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextField Example");
        JTextField textField = new JTextField("Type here");
        frame.add(textField);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JButton

- **Purpose**: Represents a push button that can trigger an action when clicked.
- **Usage**: Commonly used to perform actions when the user interacts with it.
- **Example**:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class JButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JButton Example");
        JButton button = new JButton("Click Me");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button Clicked");
            }
        });
        frame.add(button);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JCheckBox

- **Purpose**: Allows the user to select or deselect an option.
- **Usage**: Used for options where multiple selections are allowed.
- **Example**:

```
import javax.swing.JCheckBox;
import javax.swing.JFrame;

public class JCheckBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JCheckBox Example");
        JCheckBox checkBox = new JCheckBox("Check Me");
        frame.add(checkBox);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JRadioButton

- **Purpose**: Allows the user to select one option from a group of options.
- **Usage**: Used when only one option should be selected from a group.
- **Example**:

```
import javax.swing.JFrame;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JPanel;

public class JRadioButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JRadioButton Example");
        JPanel panel = new JPanel();
        JRadioButton rb1 = new JRadioButton("Option 1");
        JRadioButton rb2 = new JRadioButton("Option 2");
        ButtonGroup group = new ButtonGroup();
        group.add(rb1);
        group.add(rb2);
        panel.add(rb1);
        panel.add(rb2);
        frame.add(panel);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
```

```
    }
}
```

JComboBox

- **Purpose**: Allows the user to choose from a drop-down list of options.
- **Usage**: Used for selecting one option from a predefined list.
- **Example**:

```
import javax.swing.JComboBox;
import javax.swing.JFrame;

public class JComboBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JComboBox Example");
        String[] options = {"Option 1", "Option 2", "Option 3"};
        JComboBox<String> comboBox = new JComboBox<>(options);
        frame.add(comboBox);
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JTabbedPane

- **Purpose**: Allows the user to switch between different panels using tabs.
- **Usage**: Used for organizing multiple panels within a single window.
- **Example**:

```
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
import javax.swing.JPanel;
import javax.swing.JLabel;

public class JTabbedPaneExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTabbedPane Example");
        JTabbedPane tabbedPane = new JTabbedPane();
        JPanel panel1 = new JPanel();
        panel1.add(new JLabel("Panel 1"));
        JPanel panel2 = new JPanel();
        panel2.add(new JLabel("Panel 2"));
```

```
        tabbedPane.addTab("Tab 1", panel1);
        tabbedPane.addTab("Tab 2", panel2);
        frame.add(tabbedPane);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JScrollPane

- **Purpose**: Provides a scrollable view of another component.
- **Usage**: Used to make components like `JTextArea` or `JTable` scrollable.
- **Example**:

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JFrame;

public class JScrollPaneExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JScrollPane Example");
        JTextArea textArea = new JTextArea(10, 30);
        JScrollPane scrollPane = new JScrollPane(textArea);
        frame.add(scrollPane);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

JTree

- **Purpose**: Displays hierarchical data in a tree structure.
- **Usage**: Used for representing data in a tree format, such as file systems or organizational structures.
- **Example**:

```
import javax.swing.JFrame;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
```

```
public class JTreeExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTree Example");
        DefaultMutableTreeNode root = new
DefaultMutableTreeNode("Root");
        DefaultMutableTreeNode child1 = new
DefaultMutableTreeNode("Child
```

```
1");
DefaultMutableTreeNode child2 = new DefaultMutableTreeNode("Child 2");
root.add(child1);
root.add(child2);
JTree tree = new JTree(root);
frame.add(tree);
frame.setSize(300, 200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
}
```

JTable

- **Purpose**: Displays tabular data in a grid.
- **Usage**: Used to show data in a table format, allowing for easy data manipulation and viewing.
- **Example**:

```
import javax.swing.JFrame;
import javax.swing.JTable;
import javax.swing.JScrollPane;

public class JTableExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTable Example");
        String[] columns = {"Name", "Age"};
        Object[][] data = {
            {"Alice", 30},
            {"Bob", 25}
        };
        JTable table = new JTable(data, columns);
        frame.add(new JScrollPane(table));
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
            frame.setVisible(true);
        }
    }
```