Unit-4 Menus and Dialog Box

Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

```
public class MenuItem extends MenuComponent implements Accessible
```

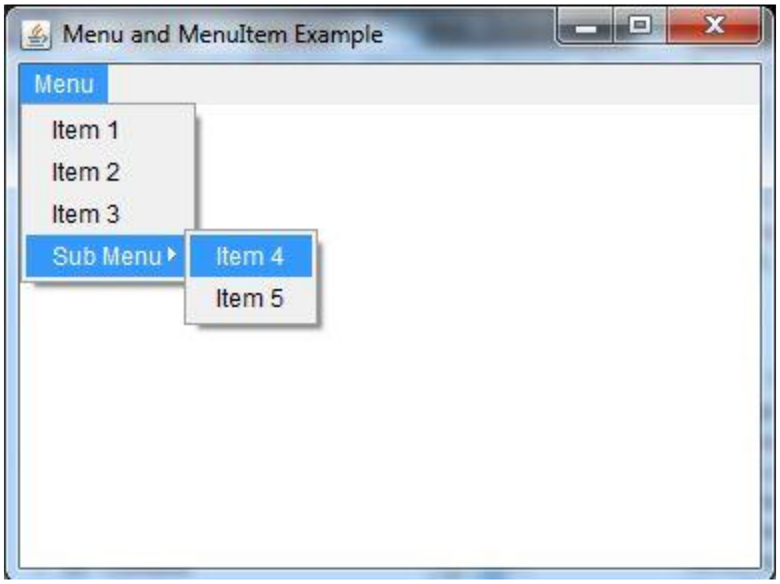
AWT Menu class declaration

```
public class Menu extends MenuItem implements MenuContainer, Accessible
```

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



Java AWT PopupMenu & Menuevents

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

AWT PopupMenu class declaration

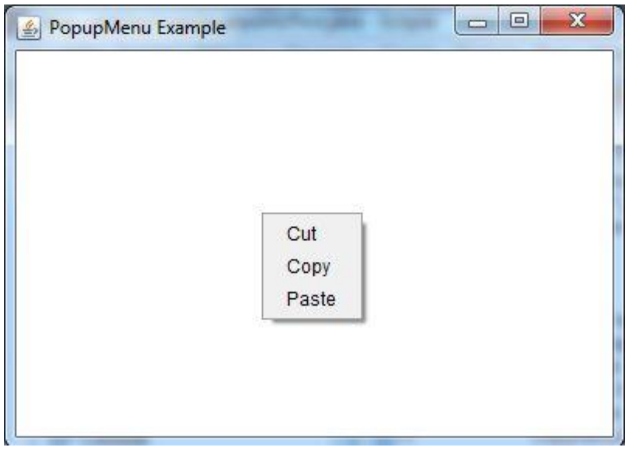
1. public class PopupMenu extends Menu implements MenuContainer, Accessible

Java AWT PopupMenu Example

```
import java.awt.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample(){
        final Frame f= new Frame("PopupMenu Example");
```

```
            final PopupMenu popupmenu = new PopupMenu("Edit");
            MenuItem cut = new MenuItem("Cut");
            cut.setActionCommand("Cut");
            MenuItem copy = new MenuItem("Copy");
            copy.setActionCommand("Copy");
            MenuItem paste = new MenuItem("Paste");
            paste.setActionCommand("Paste");
            popupmenu.add(cut);
            popupmenu.add(copy);
            popupmenu.add(paste);
            f.addMouseListener(new MouseAdapter() {
              public void mouseClicked(MouseEvent e) {
                 popupmenu.show(f , e.getX(), e.getY());
              }
            });
            f.add(popupmenu);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
          }
      public static void main(String args[])
      {
            new PopupMenuExample();
      }
      }
```

Output:



Reacting to menu events in java

In Java, reacting to **menu events** typically involves adding **action listeners** to menu items in a JMenu. The listener listens for user interactions (such as a click or a keyboard shortcut) and triggers a specific action.

Let's break down how you can set up an event listener for menu items and handle actions.

Steps:

1. **Create a menu bar** (JMenuBar) and a menu (JMenu).
2. **Add menu items** (JMenuItem) to the menu.
3. **Attach action listeners** to the menu items so they can react to user events.
4. **Define actions** (what should happen when a user selects a menu item).

Example: Reacting to Menu Events

In this example, we'll create a simple GUI with a **File** menu containing:

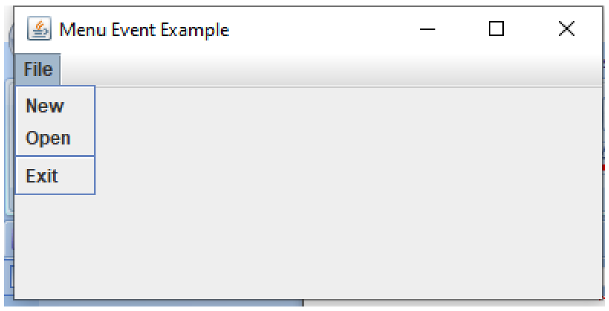
- **New** (creates a new file, triggered by a menu item)
- **Open** (opens a file, triggered by a menu item)
- **Exit** (closes the application, triggered by a menu item)

Example:

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

public class MenuEventExample {

  public static void main(String[] args) {
    // Create the main frame
    JFrame frame = new JFrame("Menu Event Example");

    // Create a menu bar
    JMenuBar menuBar = new JMenuBar();

    // Create the "File" menu
    JMenu fileMenu = new JMenu("File");

    // Create "New" menu item
    JMenuItem newItem = new JMenuItem("New");
    newItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        // React to "New" action
```

```
                System.out.println("New file created.");
            }
        });

        // Create "Open" menu item
        JMenuItem openItem = new JMenuItem("Open");
        openItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // React to "Open" action
                System.out.println("Open file dialog.");
            }
        });

        // Create "Exit" menu item
        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // React to "Exit" action
                System.out.println("Exiting application...");
                System.exit(0); // Exits the application
            }
        });

        // Add the menu items to the File menu
        fileMenu.add(newItem);
        fileMenu.add(openItem);
        fileMenu.addSeparator();  // Adds a separator between items
        fileMenu.add(exitItem);

        // Add the File menu to the menu bar
        menuBar.add(fileMenu);

        // Set the menu bar to the frame
        frame.setJMenuBar(menuBar);

        // Basic setup for the frame
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Output:



In this example, the event handling for the menu items allows the program to respond to user interactions. You can expand this to more complex actions, such as opening file dialogs, saving files, or performing other operations depending on your application's requirements.

Checkbox menuitem in java

A **checkbox menu item** in Java is a special type of menu item that can either be checked or unchecked, allowing users to toggle its state. It's commonly used for options that can be turned on or off (such as enabling or disabling features). Java provides the JCheckBoxMenuItem class to create checkbox menu items.

How to Use JCheckBoxMenuItem:

- A JCheckBoxMenuItem behaves like a normal menu item, but it has a checked or unchecked state that toggles when the user clicks on it.
- You can add an ActionListener to react to the state change of the checkbox.

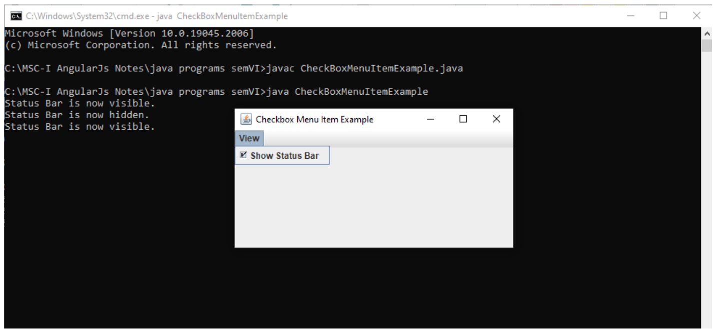
Example: Using JCheckBoxMenuItem

In this example, we'll create a simple menu with a **"Show Status Bar"** checkbox menu item that toggles its state. If checked, we'll simulate showing a status bar; if unchecked, we'll simulate hiding it.

Example:

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

public class CheckBoxMenuItemExample {

  public static void main(String[] args) {
    // Create the main frame
    JFrame frame = new JFrame("Checkbox Menu Item Example");
```

```
        // Create a menu bar
        JMenuBar menuBar = new JMenuBar();

        // Create the "View" menu
        JMenu viewMenu = new JMenu("View");

        // Create a JCheckBoxMenuItem for showing/hiding the status bar
        JCheckBoxMenuItem statusBarItem = new JCheckBoxMenuItem("Show Status Bar");

        // Set the default state of the checkbox (unchecked by default)
        statusBarItem.setSelected(false);

        // Add an ActionListener to the checkbox menu item
        statusBarItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // React to the change in the checkbox state
                if (statusBarItem.isSelected()) {
                    System.out.println("Status Bar is now visible.");
                } else {
                    System.out.println("Status Bar is now hidden.");
                }
            }
        });

        // Add the checkbox item to the View menu
        viewMenu.add(statusBarItem);

        // Add the View menu to the menu bar
        menuBar.add(viewMenu);

        // Set the menu bar to the frame
        frame.setJMenuBar(menuBar);

        // Basic setup for the frame
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```
Output:


C:\Windows\System32\cmd.exe - java CheckBoxMenuItemExample
Checkbox Menu Item Example
View
Show Status Bar

```
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\MSC-I AngularJs Notes\java programs semVI>javac CheckBoxMenuItemExample.java

C:\MSC-I AngularJs Notes\java programs semVI>java CheckBoxMenuItemExample
Status Bar is now visible.
Status Bar is now hidden.
Status Bar is now visible.
```

radiobutton menu item in java

A **RadioButtonMenuItem** in Java is used to create a menu item that behaves like a radio button in a menu. It's part of a group of menu items where only one item can be selected at a time. This is useful for creating mutually exclusive options, such as selecting a mode or setting where only one option can be active at any given time.

Java provides the JRadioButtonMenuItem class, which works similarly to a normal JMenuItem, but it can be grouped with other JRadioButtonMenuItems to ensure that only one of them can be selected at a time.

How to Use JRadioButtonMenuItem:

- JRadioButtonMenuItem behaves like a radio button, so when one is selected, all others in the same group are automatically deselected.
- Grouping JRadioButtonMenuItem instances together ensures that only one can be selected at any time. This is done using ButtonGroup.

Example: Using JRadioButtonMenuItem

Let’s create a simple example with a **Theme** menu that allows users to select between three different themes (Light, Dark, and Blue). Only one theme can be selected at a time.

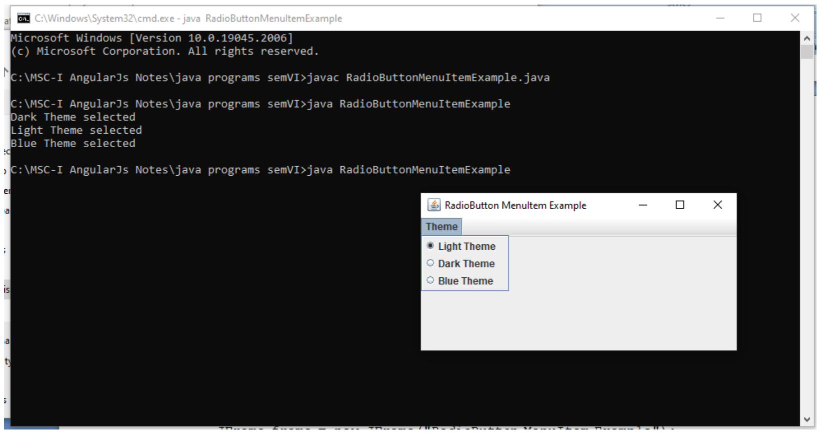
Example:

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

public class RadioButtonMenuItemExample {

   public static void main(String[] args) {
      // Create the main frame
      JFrame frame = new JFrame("RadioButton MenuItem Example");
```

```
// Create a menu bar
JMenuBar menuBar = new JMenuBar();

// Create the "Theme" menu
JMenu themeMenu = new JMenu("Theme");

// Create JRadioButtonMenuItems for different themes
JRadioButtonMenuItem lightThemeItem = new JRadioButtonMenuItem("Light Theme");
JRadioButtonMenuItem darkThemeItem = new JRadioButtonMenuItem("Dark Theme");
JRadioButtonMenuItem blueThemeItem = new JRadioButtonMenuItem("Blue Theme");

// Group the radio buttons together so only one can be selected at a time
ButtonGroup themeGroup = new ButtonGroup();
themeGroup.add(lightThemeItem);
themeGroup.add(darkThemeItem);
themeGroup.add(blueThemeItem);

// Add action listeners to react to theme selection
lightThemeItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Light Theme selected");
    }
});

darkThemeItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Dark Theme selected");
    }
});

blueThemeItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Blue Theme selected");
    }
});

// Set the default selected theme (for example, Light Theme)
lightThemeItem.setSelected(true);

// Add the radio button menu items to the Theme menu
themeMenu.add(lightThemeItem);
themeMenu.add(darkThemeItem);
themeMenu.add(blueThemeItem);
```

```
        // Add the Theme menu to the menu bar
        menuBar.add(themeMenu);

        // Set the menu bar to the frame
        frame.setJMenuBar(menuBar);

        // Basic setup for the frame
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Output:



```
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\MSC-I AngularJs Notes\java programs semVI>javac RadioButtonMenuItemExample.java

C:\MSC-I AngularJs Notes\java programs semVI>java RadioButtonMenuItemExample
Dark Theme selected
Light Theme selected
Blue Theme selected

C:\MSC-I AngularJs Notes\java programs semVI>java RadioButtonMenuItemExample
```

C:\Windows\System32\cmd.exe - java RadioButtonMenuItemExample
RadioButton MenuItem Example
Theme
Light Theme
Dark Theme
Blue Theme

Keyboard mnemonics and Accelerators:

A **mnemonic** allows you to use keyboard to select a menu item or submenu from a menu that is already open.

An **accelerator** is a key that lets you select a menu item without ever opening a menu.

In Java, keyboard mnemonics and accelerators are used to improve the user experience by enabling keyboard shortcuts for interacting with graphical user interface (GUI) components, such as menus and buttons.

1. **Keyboard Mnemonics**:

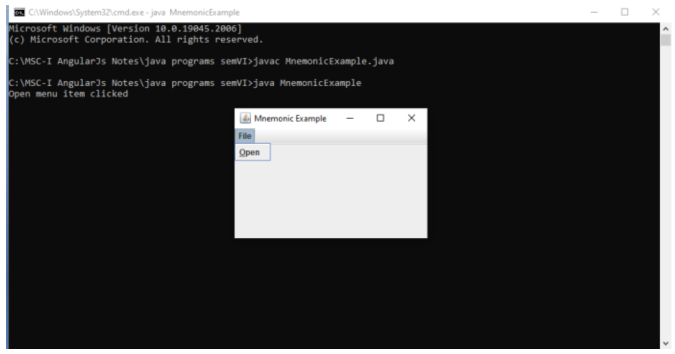
A **mnemonic** is a key or combination of keys that the user can press to activate a specific component (such as a button or a menu item) on a GUI, typically by underlining the corresponding character.

In Java, you can set a mnemonic for a GUI component using the setMnemonic() method for menu items or buttons.

Example:

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

public class MnemonicExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Mnemonic Example");
      JMenuBar menuBar = new JMenuBar();
      JMenu menu = new JMenu("File");

      // Creating a menu item with mnemonic 'O' for Open
      JMenuItem openItem = new JMenuItem("Open");
      openItem.setMnemonic(KeyEvent.VK_O);

      // Adding action listener for the menu item
      openItem.addActionListener(e -> System.out.println("Open menu item clicked"));

      menu.add(openItem);
      menuBar.add(menu);
      frame.setJMenuBar(menuBar);
      frame.setSize(300, 200);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Output:



```
C:\Windows\System32\cmd.exe - java  MnemonicExample
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\MSC-I AngularJs Notes\java programs semVI>javac MnemonicExample.java

C:\MSC-I AngularJs Notes\java programs semVI>java MnemonicExample
Open menu item clicked
```

- In the above example, the **Open** menu item has a mnemonic O. When the user presses Alt + O (or the corresponding key depending on the OS), it will trigger the action associated with that menu item.

2. Keyboard Accelerators**:**

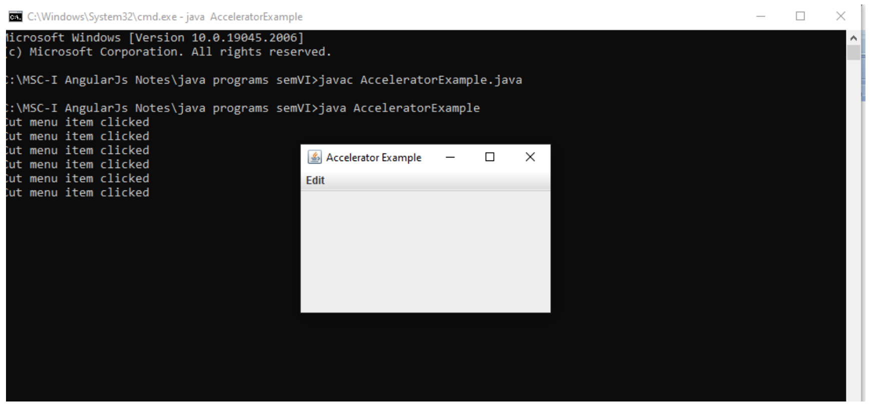
An **accelerator** is a more general shortcut that works regardless of focus and is typically associated with a specific action (e.g., Ctrl + S for saving).

In Java, you can set an accelerator for a menu item using the setAccelerator() method.

Example:

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

public class AcceleratorExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Accelerator Example");
        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu("Edit");

        // Creating a menu item for Cut with accelerator Ctrl+X
        JMenuItem cutItem = new JMenuItem("Cut");
        cutItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X,
InputEvent.CTRL_DOWN_MASK));

        // Adding action listener for the menu item
        cutItem.addActionListener(e -> System.out.println("Cut menu item clicked"));

        menu.add(cutItem);
        menuBar.add(menu);
        frame.setJMenuBar(menuBar);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Output:



```
C:\Windows\System32\cmd.exe - java AcceleratorExample
icrosoft Windows [Version 10.0.19045.2006]
c) Microsoft Corporation. All rights reserved.

:\MSC-I AngularJs Notes\java programs semVI>javac AcceleratorExample.java

:\MSC-I AngularJs Notes\java programs semVI>java AcceleratorExample
ut menu item clicked
ut menu item clicked
ut menu item clicked
ut menu item clicked
ut menu item clicked
ut menu item clicked
```

- In this example, the **Cut** menu item has an accelerator `Ctrl + X`. When the user presses this key combination, it will trigger the associated action.

Mnemonic vs Accelerator:

- **Mnemonic**: Focused on triggering a specific action when the user presses a single key or combination (often with Alt), usually for GUI components like buttons or menu items.
- **Accelerator**: A key combination that invokes an action directly, often associated with commonly used tasks (like `Ctrl + S` for Save).

Dialog Boxes:

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons

.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

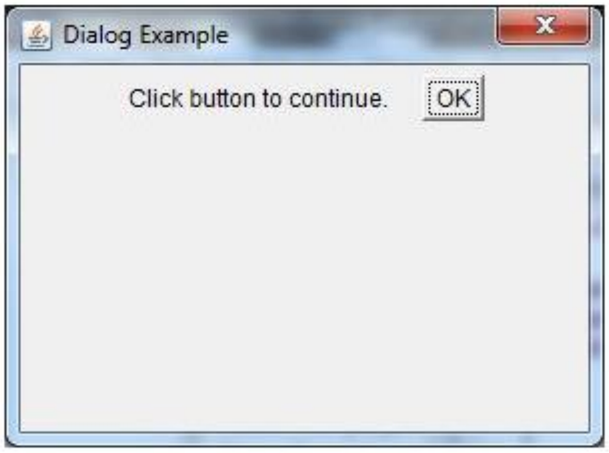
AWT Dialog class declaration

1. **public class** Dialog **extends** Window

Java AWT Dialog Example

```
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
  static   Dialog d;
   DialogExample() {
      Frame f= new Frame();
      d = new Dialog(f , "Dialog Example", true);
      d.setLayout( new FlowLayout() );
      Button b = new Button ("OK");
      b.addActionListener ( new ActionListener()
      {
         public void actionPerformed( ActionEvent e )
         {
            DialogExample.d.setVisible(false);
         }
      });
      d.add( new Label ("Click button to continue."));
      d.add(b);
      d.setSize(300,300);
      d.setVisible(true);
   }
   public static void main(String args[])
   {
      new DialogExample();
   }
}
```

Output:



Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane Class Declaration

public class JOptionPane **extends** JComponent **implements** Accessible

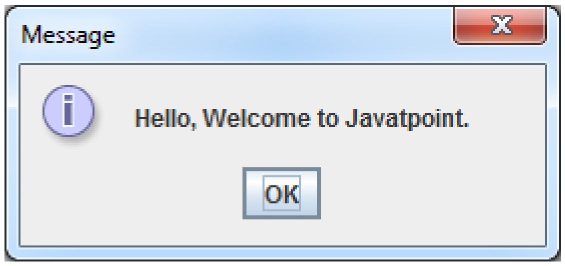
Java JOptionPane Example: showMessageDialog()

You can display simple messages in a pop-up dialog box, for example, an informational message or an error message.

Filename: OptionPaneExample.java

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Output:



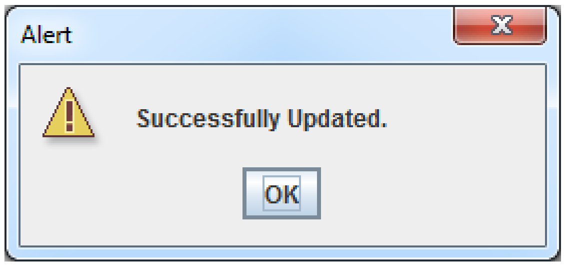
Hello, Welcome to Javatpoint.

Java JOptionPane Example: showMessageDialog()

Filename: OptionPaneExample.java

```
import javax.swing.*;
public class OptionPaneExample {

JFrame f;
OptionPaneExample(){
   f=new JFrame();
   JOptionPane.showMessageDialog(f,"Successfully
Updated.","Alert",JOptionPane.WARNING_MESSAGE);
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Output:

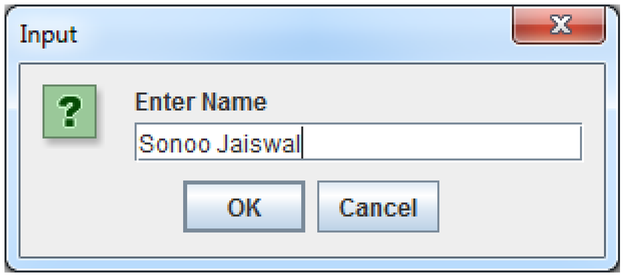


Java JOptionPane Example: showInputDialog()

Filename: OptionPaneExample.java

```
import javax.swing.*;

public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Output:

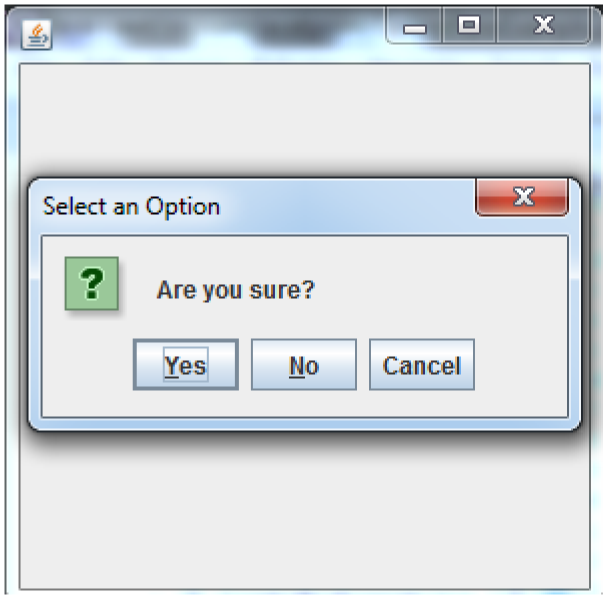


Java JOptionPane Example: showConfirmDialog()

Filename: OptionPaneExample.java

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   f.addWindowListener(this);
   f.setSize(300, 300);
   f.setLayout(null);
   f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
   f.setVisible(true);
}

public void windowClosing(WindowEvent e) {
   int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
public static void main(String[] args) {
   new  OptionPaneExample();
}
}
```

Output:



File Dialogs:

In Java, you can create a file dialog using the JFileChooser class from the Swing library. This class provides a standard file chooser for the user to select files or directories. Below is an example of how to use JFileChooser to open a file dialog for selecting a file:

Example

```
import javax.swing.*;
import java.io.File;

public class FileDialogExample {
  public static void main(String[] args) {
    // Create a JFileChooser object
    JFileChooser fileChooser = new JFileChooser();

    // Optionally, set the dialog to only allow file selection (not directories)
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

    // Open the file chooser dialog
    int result = fileChooser.showOpenDialog(null);

    // Check the result of the user's action
    if (result == JFileChooser.APPROVE_OPTION) {
            // Get the selected file
            File selectedFile = fileChooser.getSelectedFile();
            System.out.println("Selected file: " + selectedFile.getAbsolutePath());
        } else {
            System.out.println("No file selected.");
        }
    }
}
```

Key Points:

- JFileChooser.showOpenDialog() is used to display a dialog that allows the user to select an existing file.
- The method showSaveDialog() can be used if you want the user to select or specify a file to save to.
- The dialog returns an integer value that indicates the user's action (e.g., approval or cancellation).
- You can use fileChooser.getSelectedFile() to get the File object representing the chosen file.

Methods of JFileChooser:

1. **showOpenDialog(Component parent):** Displays the file dialog for opening a file.
2. **showSaveDialog(Component parent):** Displays the file dialog for saving a file.
3. **setFileSelectionMode(int mode):** Controls whether the dialog lets the user select files, directories, or both. You can set it to JFileChooser.FILES_ONLY or JFileChooser.DIRECTORIES_ONLY.