JAVA PROGRAMMING

Week 7: Swing (part 2)

Lecturer:

- HO Tuan Thanh, M.Sc.

Plan

1. **Menu**
2. **Dialog window**
3. **Other components: JFileChooser, JColorChooser**

Plan

1. **Menu**
2. **Dialog window**
3. **Other components: JFileChooser, JColorChooser**

Core Menu classes

Class	Description
JMenuBar	An object that holds the top-level menu for the application.
JMenu	A standard menu. A menu consists of one or more **JMenuItem**s.
JMenuItem	An object that populates menus.
JCheckBoxMenuItem	A check box menu item.
JRadioButtonMenuItem	A radio button menu item
JSeparator	The visual separator between menu items.
JPopupMenu	A menu that is typically activated by right-clicking the mouse.

JMenuBar, JMenu, and JMenuItem

- These form the minimum set of classes needed to construct a main menu for an application.
- JMenu and JMenuItem are also used by popup menus → the foundation of the menu system.

JMenuBar [1]

- Is essentially a container for menus.
- Inherits JComponent
- Has only one constructor: default constructor.

→Initially the menu bar will be empty

→You need to populate it with menus prior to use.

→ Each application has one and only one menu bar.

- Methods:
  - JMenu add(JMenu menu)
  - Component add(Component menu, int idx)
  - void remove(Component menu)
  - void remove(int idx)

JMenuBar [2]

- Methods (cont.)
  - int getMenuCount()
  - MenuElement[] getSubElements()
  - boolean isSelected()
  - void setJMenuBar(JMenuBar mb)

JMenu [1]

- JMenu encapsulates a menu, which is populated with JMenuItems.
- It is derived from JMenuItem → one JMenu can be a selection in another JMenu → one menu to be a submenu of another.
- Constructors:
  - JMenu(String name)
  - JMenu()
- Methods
  - JMenuItem add(JMenuItem item)
  - Component add(Component item, int idx)
  - void addSeparator()

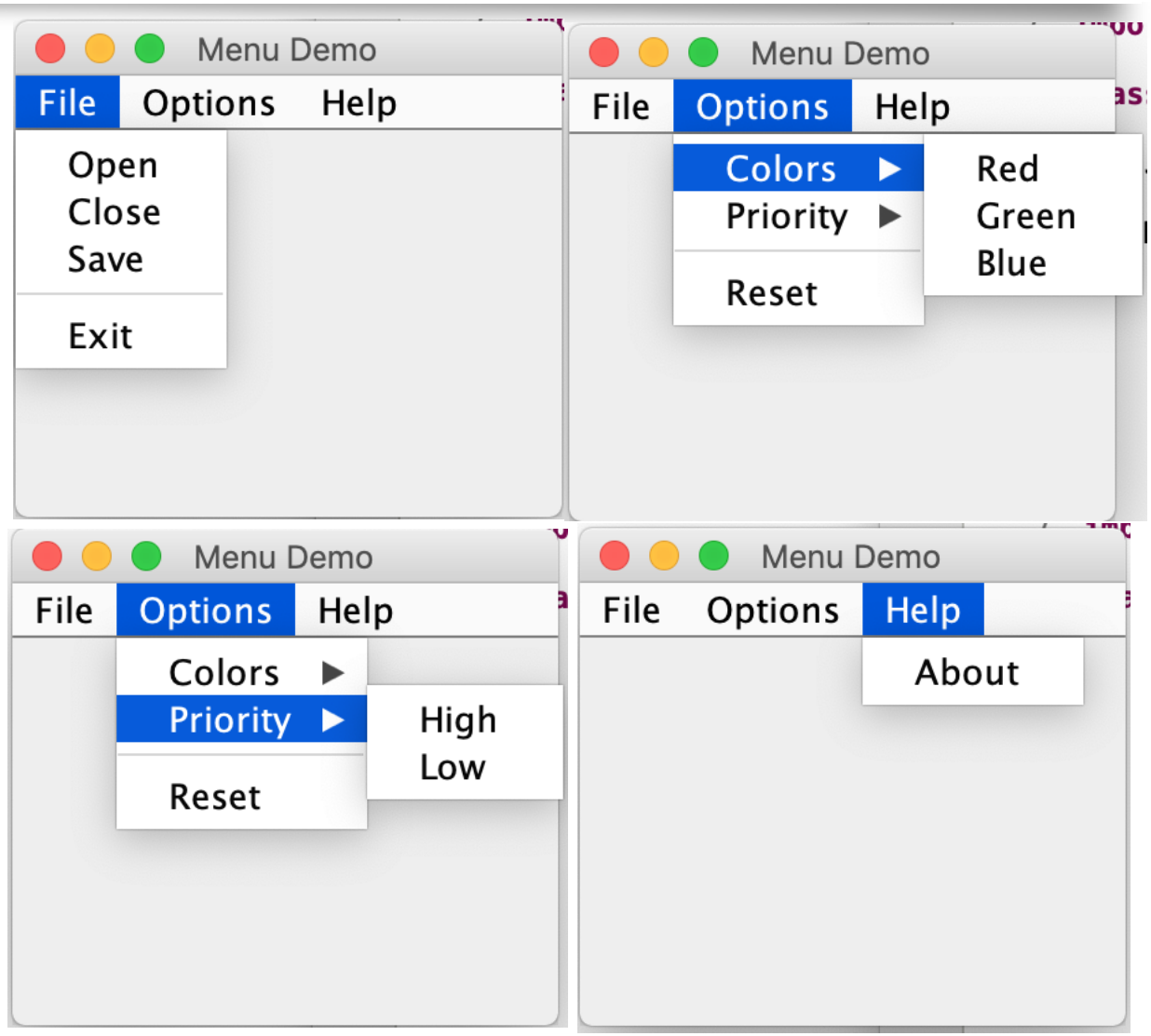
JMenu [2]

- Methods
  - void insertSeparator(int idx)
  - void remove(JMenuItem menu)
  - void remove(int idx)
  - int getMenuComponentCount()
  - Component[ ] getMenuComponents()

JMenuItem

- JMenuItem encapsulates an element in a menu.
- JMenuItem is derived from AbstractButton, and every item in a menu can be thought of as a special kind of button → when a menu item is selected, an action event is generated.
- Constructors:
  - JMenuItem(String name)
  - JMenuItem(Icon image)
  - JMenuItem(String name, Icon image)
  - JMenuItem(String name, int mnem)
  - JMenuItem(Action action)
- Methods
  - void setEnabled(boolean enable)

Example: Creation of main menu



```
1.  //Demonstrate a simple main menu.
2.  import java.awt.*;
3.  import java.awt.event.*;
4.  import javax.swing.*;
5.  class MenuDemo implements ActionListener {
6.          JLabel jlab;
7.          MenuDemo() {
8.                  // Create a new JFrame container.
9.                  JFrame jfrm = new JFrame("Menu Demo");
10.                 // Specify FlowLayout for the layout manager.
11.                 jfrm.setLayout(new FlowLayout());
12.                 // Give the frame an initial size.
13.                 jfrm.setSize(220, 200);
14.                 //Terminate the program when the user closes the application.
15.                 jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16.                 // Create a label that will display the menu selection.
17.                 jlab = new JLabel();
18.                 // Create the menu bar.
19.                 JMenuBar jmb = new JMenuBar();
20.
```

```
// Create the File menu.
JMenu jmFile = new JMenu("File");
JMenuItem jmiOpen = new JMenuItem("Open");
JMenuItem jmiClose = new JMenuItem("Close");
JMenuItem jmiSave = new JMenuItem("Save");
JMenuItem jmiExit = new JMenuItem("Exit");
jmFile.add(jmiOpen);
jmFile.add(jmiClose);
jmFile.add(jmiSave);
jmFile.addSeparator();
jmFile.add(jmiExit);
jmb.add(jmFile);
```

```
// Create the Options menu.
JMenu jmOptions = new JMenu("Options");
// Create the Colors submenu.
JMenu jmColors = new JMenu("Colors");
JMenuItem jmiRed = new JMenuItem("Red");
JMenuItem jmiGreen = new JMenuItem("Green");
JMenuItem jmiBlue = new JMenuItem("Blue");
jmColors.add(jmiRed);
jmColors.add(jmiGreen);
jmColors.add(jmiBlue);
jmOptions.add(jmColors);
// Create the Priority submenu.
JMenu jmPriority = new JMenu("Priority");
JMenuItem jmiHigh = new JMenuItem("High");
JMenuItem jmiLow = new JMenuItem("Low");
jmPriority.add(jmiHigh);
jmPriority.add(jmiLow);
jmOptions.add(jmPriority);
```

```
// Create the Reset menu item.
JMenuItem jmiReset = new JMenuItem("Reset");
jmOptions.addSeparator(); jmOptions.add(jmiReset);
// Finally, add the entire options menu to the menu bar
jmb.add(jmOptions);
// Create the Help menu.
JMenu jmHelp = new JMenu("Help");
JMenuItem jmiAbout = new JMenuItem("About");
jmHelp.add(jmiAbout); jmb.add(jmHelp);
// Add action listeners for the menu items.
jmiOpen.addActionListener(this);
jmiClose.addActionListener(this);
jmiSave.addActionListener(this);
jmiExit.addActionListener(this);
jmiRed.addActionListener(this);
jmiGreen.addActionListener(this);
jmiBlue.addActionListener(this);
jmiHigh.addActionListener(this);
jmiLow.addActionListener(this);
jmiReset.addActionListener(this);
jmiAbout.addActionListener(this);
```

```
1.          jfrm.add(jlab); // Add the label to the content pane.
2.          // Add the menu bar to the frame.
3.          jfrm.setJMenuBar(jmb);
4.          jfrm.setVisible(true); // Display the frame.
5.      }
6.      //Handle menu item action events.
7.      public void actionPerformed(ActionEvent ae) {
8.          // Get the action command from the menu selection.
9.          String comStr = ae.getActionCommand();
10.         // If user chooses Exit, then exit the program.
11.         if (comStr.equals("Exit")) System.exit(0);
12.         // Otherwise, display the selection.
13.         jlab.setText(comStr + " Selected");
14.     }
15.     public static void main(String args[]) {
16.         // Create the frame on the event dispatching thread.
17.         SwingUtilities.invokeLater(new Runnable() {
18.             public void run() { new MenuDemo(); }
19.         });
20.     }
21. }
```

Add Mnemonics and Accelerators to Menu Items

- In real applications: a menu usually includes support for keyboard shortcuts → ability to select menu items rapidly.
- Two forms:
  - mnemonics and
  - accelerators.
- a mnemonic allows you to use the keyboard to select an item from a menu that is already being displayed.
- An accelerator is a key that lets you select a menu item without having to first activate the menu.

mnemonic [1]

- A mnemonic can be specified for both JMenuItem and JMenu objects.
- Two ways to set the mnemonic for JMenuItem
  - Using constructor : JMenuItem (String name, int mnem)
  - Set the mnemonic : void setMnemonic (int mnem)
  - mnem specifies the mnemonic. It should be one of the constants defined in java.awt.event.KeyEvent, such as KeyEvent.VK_F ou KeyEvent.VK_Z.
  - Mnemonics are not case sensitive.

mnemonic [2]

- By default: the first matching letter in the menu item will be underscored.
  - In cases in which you want to underscore a letter other than the first match:

    void setDisplayedMnemonicIndex(int idx)

  - The index of the letter to underscore is specified by idx.

Accelerator [1]

- An accelerator can be associated with a JMenuItem object.
- It is specified by calling

  void setAccelerator (KeyStroke ks)

  - ks is the key combination that is pressed to select the menu item.
  - KeyStroke is a class that contains several factory methods that construct various types of keystroke accelerators.
- Example:
  - static KeyStroke getKeyStroke(char ch)
  - static KeyStroke getKeyStroke(Character ch, int modifier)
  - static KeyStroke getKeyStroke(int ch, int modifier)
  - ch specifies the accelerator character.

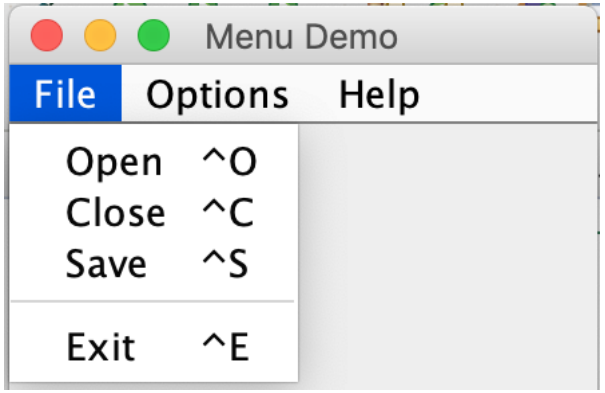
Accelerator [2]

- modifier must be one or more of the following constants, defined in the java.awt.event.InputEvent class :

InputEvent.ALT_DOWN_MASK	InputEvent.ALT_GRAPH_DOWN_MASK
InputEvent.CTRL_DOWN_MASK	InputEvent.META_DOWN_MASK
InputEvent.SHIFT_DOWN_MASK	

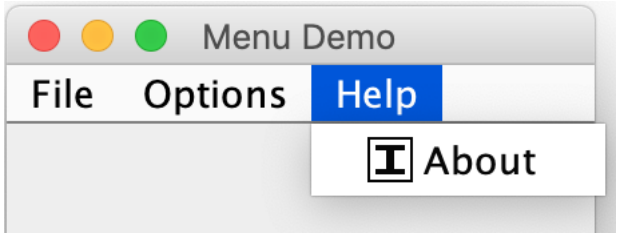
- If you pass VK_A for the key character and InputEvent.CTRL_DOWN_MASK for the modifier → the accelerator key combination is CTRL-A.

```
// Create the File menu with mnemonics and accelerators.
jmFile.setMnemonic(KeyEvent.VK_F);
JMenuItem jmiOpen = new JMenuItem("Open", KeyEvent.VK_O);
jmiOpen.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
                                                InputEvent.CTRL_DOWN_MASK));
JMenuItem jmiClose = new JMenuItem("Close", KeyEvent.VK_C);
jmiClose.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
                                                InputEvent.CTRL_DOWN_MASK));
JMenuItem jmiSave = new JMenuItem("Save", KeyEvent.VK_S);
jmiSave.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
                                                InputEvent.CTRL_DOWN_MASK));
JMenuItem jmiExit = new JMenuItem("Exit", KeyEvent.VK_E);
jmiExit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
                                                InputEvent.CTRL_DOWN_MASK));
```



Add Images and Tooltips to Menu Items [1]

- You can add images to menu items or use images instead of text.
- Constructors:
  - JMenuItem(Icon image)
  - JMenuItem(String name, Icon image)
- Example:
  - ImageIcon icon = new ImageIcon("AboutIcon.gif");
  - JMenuItem jmiAbout = new JMenuItem("About", icon);



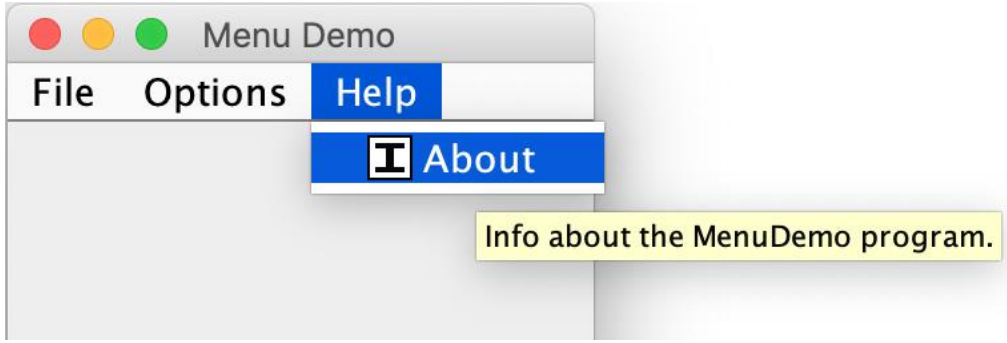
Add Images and Tooltips to Menu Items [2]

- To add an icon to a menu item (inherit from AbstractButton)
  - void setIcon(Icon defaultIcon)
- To specify horizontal alignment of image:
  - void setHorizontalAlignment(int alignment)
- To specify a disabled icon:
  - void setDisabledIcon(Icon disabledIcon)

Tooltip

- A tooltip is a small message that describes an item.
- It is automatically displayed if the mouse remains over the item for a moment.
- To add a tooltip to a menu item:
  - void setToolTipText(String msg)
- Example

```
jmiAbout.setToolTipText
                    ("Info about the MenuDemo program.");
```

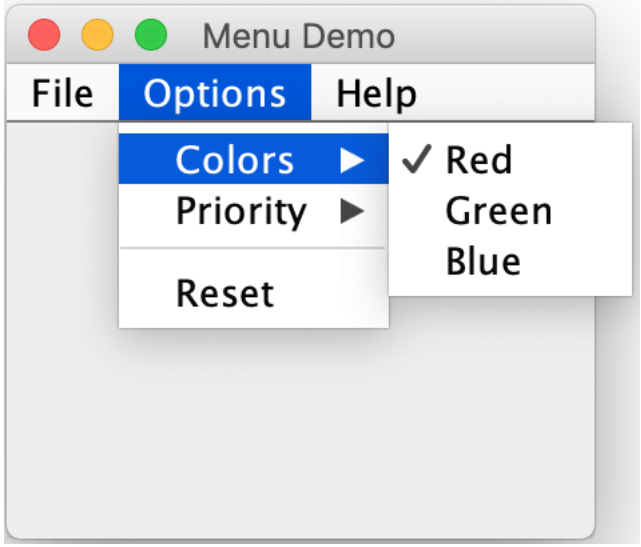


JRadioButtonMenuItem and JCheckBoxMenuItem

- Menu items are the most commonly used.
- Swing defines two others: check boxes and radio buttons.
  - Allow a menu to provide functionality that would otherwise require additional, standalone components.
  - Sometimes, including check boxes or radio buttons in a menu simply seems the most natural place for a specific set of features.
- To add a check box to a menu: create a JCheckBoxMenuItem.
- Constructors :
  - JCheckBoxMenuItem(String name)
  - JCheckBoxMenuItem(String name, boolean state)
  - JCheckBoxMenuItem(String name, Icon icon)

Example: JCheckBoxMenuItem

```
// Use check boxes for colors. This allows
// the user to select more than one color.
JCheckBoxMenuItem jmiRed = new JCheckBoxMenuItem("Red", true);
JCheckBoxMenuItem jmiGreen = new JCheckBoxMenuItem("Green", false);
JCheckBoxMenuItem jmiBlue = new JCheckBoxMenuItem("Blue", false);
```



JRadioButtonMenuItem

- Constructors
  - JRadioButtonMenuItem(String name)
  - JRadioButtonMenuItem(String name, boolean state)
  - JRadioButtonMenuItem(String name, Icon icon, boolean state)
- Example

```
JRadioButtonMenuItem jmiHigh =
                    new JRadioButtonMenuItem("High", true);
JRadioButtonMenuItem jmiLow =
                    new JRadioButtonMenuItem("Low");
// Create button group for the radio button menu items.
ButtonGroup bg = new ButtonGroup();
bg.add(jmiHigh); bg.add(jmiLow);
```

Create a Popup Menu [1]

- A popular alternative or addition to the menu bar.
- A popup menu is activated by clicking the right mouse button when over a component.
- Constructors
  - JPopupMenu()
  - JPopupMenu(String label)
- Popup menus are constructed like regular menus
  - First, create a JPopupMenu object, and then add menu items to it.
  - Menu item selections are also handled in the same way: by listening for action events.

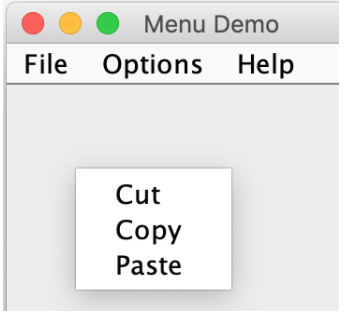
Create a Popup Menu [2]

- The main difference between a popup menu and regular menu: the activation process.
  1. You must register a listener for mouse events.
  2. Inside the mouse event handler, you must watch for the popup trigger.
  3. When a popup trigger is received, you must show the popup menu by calling `show( )`.
- To listen for the popup trigger:
  - Implement the `MouseListener` interface and
  - Register the listener by calling the `addMouseListener( )` method

Create a Popup Menu [3]

- Methods of MouseListener:
  - void mouseClicked(MouseEvent me)
  - void mouseEntered(MouseEvent me)
  - void mouseExited(MouseEvent me)
  - void mousePressed(MouseEvent me)
  - void mouseReleased(MouseEvent me)
- Four methods of MouseEvent class are commonly needed when activating a popup menu:
  - int getX( )
  - int getY( )
  - boolean isPopupTrigger( )
  - Component getComponent( )

```
// Create an Edit popup menu.
JPopupMenu jpu = new JPopupMenu();
// Create the popup menu items
JMenuItem jmiCut = new JMenuItem("Cut");
JMenuItem jmiCopy = new JMenuItem("Copy");
JMenuItem jmiPaste = new JMenuItem("Paste");
// Add the menu items to the popup menu.
jpu.add(jmiCut); jpu.add(jmiCopy); jpu.add(jmiPaste);
// Add a listener for for the popup trigger.
jfrm.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent me) {
    if(me.isPopupTrigger())
      jpu.show(me.getComponent(), me.getX(), me.getY());
  }
  public void mouseReleased(MouseEvent me) {
    if(me.isPopupTrigger())
      jpu.show(me.getComponent(), me.getX(), me.getY());
  }
});
```



Toolbar

- A toolbar is a component that can serve as both an alternative and as an adjunct to a menu.
- A toolbar contains a list of buttons (or other components) that give the user immediate access to various program options.
- In general:
  - Toolbar buttons show icons rather than text, although either or both are allowed.
  - Tooltips are often associated with icon-based toolbar buttons.
- Toolbars can be positioned on any side of a window by dragging the toolbar, or they can be dragged out of the window entirely, in which case they become free floating.

Create a Toolbar [1]

- To create a toolbar: JToolBar
- Constructors
  - JToolBar ()
  - JToolBar (String title)
  - JToolBar (int how)
  - JToolBar (String title, int how)
- To add buttons (or other components) to a toolbar: Use add().

```
// Create a Debug tool bar.
JToolBar jtb = new JToolBar("Debug");
// Load the images.
ImageIcon set = new ImageIcon("setBP.gif");
ImageIcon clear = new ImageIcon("clearBP.gif");
ImageIcon resume = new ImageIcon("resume.gif");
// Create the tool bar buttons.
JButton jbtnSet = new JButton(set);
jbtnSet.setActionCommand("Set Breakpoint");
jbtnSet.setToolTipText("Set Breakpoint");
JButton jbtnClear = new JButton(clear);
jbtnClear.setActionCommand("Clear Breakpoint");
jbtnClear.setToolTipText("Clear Breakpoint");
JButton jbtnResume = new JButton(resume);
jbtnResume.setActionCommand("Resume");
jbtnResume.setToolTipText("Resume");
// Add the buttons to the toolbar.
jtb.add(jbtnSet);  jtb.add(jbtnClear);  jtb.add(jbtnResume)
jfrm.add(jtb, BorderLayout.NORTH);
```

Plan

1. Menu
2. **Dialog window**
3. Other components: JFileChooser, JColorChooser

Dialog window

- An independent subwindow meant to carry temporary notice apart from the main Swing Application Window.
- Most Dialogs present an error message or warning to a user, but Dialogs can present images, directory trees, or just about anything compatible with the main Swing Application that manages them.
- Several Swing component classes can directly instantiate and display dialogs.

Examples of Dialog windows

- To create simple, standard dialogs: use the JOptionPane class.
- The ProgressMonitor class can put up a dialog that shows the progress of an operation.
- JColorChooser and JFileChooser supply standard dialogs.
- To bring up a print dialog, you can use the Printing API.
- To create a custom dialog, use the JDialog class directly.

Overview of Dialogs [1]

- Every dialog is dependent on a `Frame` component.
  - When that `Frame` is destroyed: its dependent Dialogs are destroyed, too.
  - When the frame is iconified: its dependent Dialogs also disappear from the screen.
  - When the frame is deiconified: its dependent Dialogs return to the screen.
  - A Swing `JDialog` class inherits this behavior from the AWT Dialog class.
- A Dialog can be *modal*.
  - `JOptionPane` creates JDialogs that are modal.
  - To create a non-modal Dialog: use the JDialog class directly.

Overview of Dialogs [2]

- From JDK 7: you can modify dialog window modality behavior using the new Modality API. For more information: click HERE.
- The JDialog class is a subclass of the AWT `java.awt.Dialog` class.
  - It adds a root pane container and support for a default close operation to the Dialog object .
- When you use `JOptionPane` to implement a dialog, you're still using a JDialog behind the scenes.
  - Reason: `JOptionPane` is simply a container that can automatically create a `JDialog` and add itself to the JDialog's content pane.

JOptionPane Features

- JOptionPane provides support for laying out standard dialogs, providing icons, specifying the dialog title and text, and customizing the button text.
- Allow you to customize the components the dialog displays and specify where the dialog should appear onscreen.
- You can even specify that an option pane put itself into an internal frame (JInternalFrame) instead of a JDialog.

Icons used by JOptionPane

Icon description	Java look and feel	Windows look and feel
question		
information		
warning		
error		

Example

```
JOptionPane.showMessageDialog(frame,
        "Eggs are not supposed to be green.");
```

Methodes of JOptionPane

- showConfirmDialog()
  - Asks a confirming question, like yes/no/cancel.
- showInputDialog()
  - Prompt for some input.
- showMessageDialog()
  - Tell the user about something that has happened.
- showOptionDialog()
  - The Grand Unification of the above three.

→ show*Xxx*Dialog

Arguments of show*Xxx*Dialog [1]

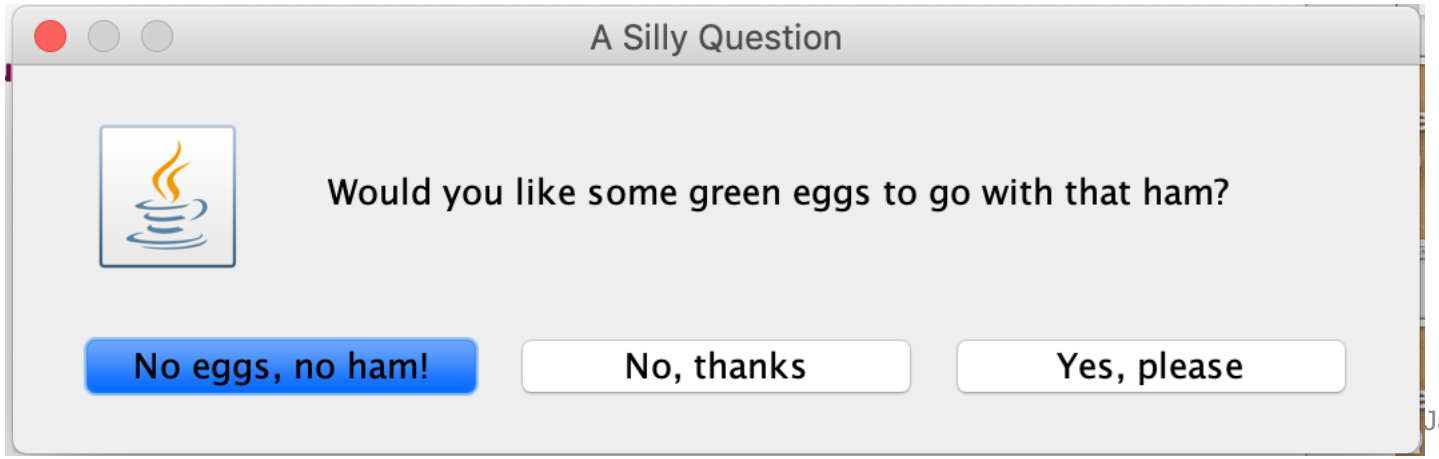
- Component parentComponent
  - the parent component, which must be a Frame, a component inside a Frame, or null.
- Object message
  - specifies what the dialog should display in its main area.
- String title
  - The title of the dialog.
- int optionType
  - Specifies the set of buttons that appear at the bottom of the dialog.
  - Values: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION.

Arguments of show*Xxx*Dialog [2]

- int messageType
  - determines the icon displayed in the dialog.
  - Values: PLAIN_MESSAGE (without icon), ERROR_MESSAGE, INFORMATION_MESSAGE,WARNING_MESSAGE, QUESTION_MESSAGE.
- Icon icon
  - The icon to display in the dialog.
- Object[] options
  - specify the string displayed by each button at the bottom of the dialog.
- Object initialValue
  - Specifies the default value to be selected.

Example

```
//Custom button text
Object[] options = {"Yes, please", "No, thanks",
                "No eggs, no ham!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like some green eggs to go with that ham?",
    "A Silly Question", JOptionPane.YES_NO_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,  null, options,
            options[2]);
```



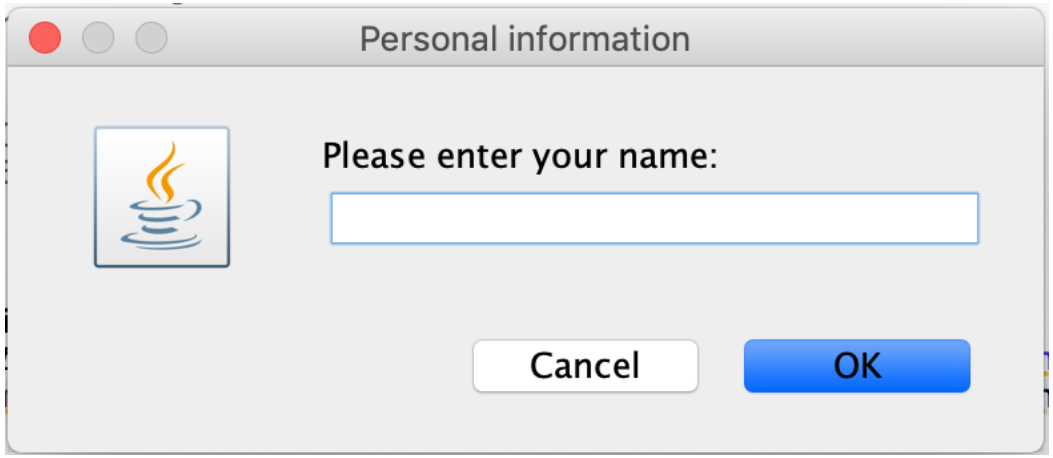
Obtaining user input from a dialog window

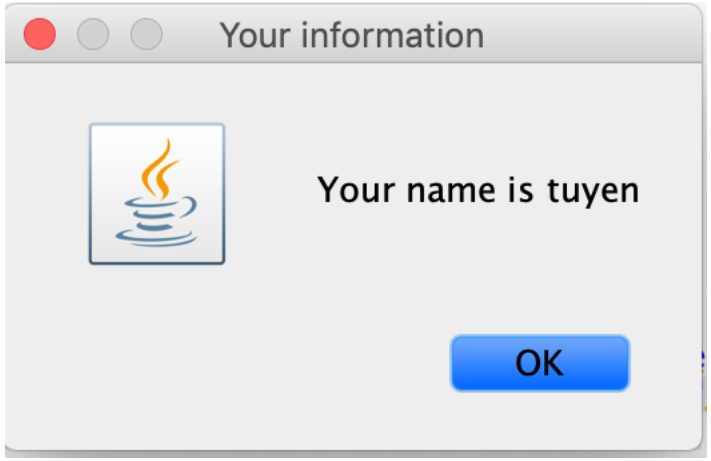
- Only showInputDialog returns an Object.
  - This Object is generally a String reflecting the user's choice.
- Example:

```
String name = JOptionPane.showInputDialog(frame,
                    "Please enter your name: ",
                    "Personal information",
                    JOptionPane.QUESTION_MESSAGE);

JOptionPane.showMessageDialog(frame, "Your name is " + name,
                    "Your information",
                    JOptionPane.INFORMATION_MESSAGE);
```

Result


Personal information
Please enter your name:
Cancel
OK


Your information
Your name is tuyen
OK

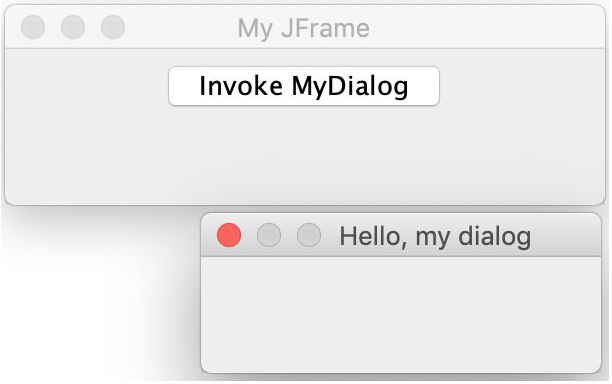
Create a custom dialog

- Create a class which extends JDialog.

Example

```
public class MyDialog extends JDialog {
    MyDialog(JFrame parent, String title, boolean modal){
        // We call the corresponding JDialog constructor
      super(parent, title, modal);
      // We specify the size of the dialog
      this.setSize(200, 80);
      // The position
      this.setLocationRelativeTo(null);
      // The box should not be resizable
      this.setResizable(false);
      // Finally we display it
      this.setVisible(true);
    }
}
```

```
1.  public class MyDialogDemo extends JFrame {
2.  private JButton button = new
3.                                  JButton("Invoke MyDialog");
4.  public MyDialogDemo() {
5.      setTitle("My JFrame");  setSize(300, 100);
6.      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7.      setLocationRelativeTo(null);
8.      getContentPane().setLayout(new FlowLayout());
9.      getContentPane().add(button);
10.     button.addActionListener(new ActionListener(){
11.         public void actionPerformed(ActionEvent ae) {
12.             new MyDialog(null, "Hello, my dialog", true);
13.         }
14.     });
15.     this.setVisible(true);
16. }
17. public static void main(String[] args){
18.     new MyDialogDemo();
19. }
```



Example (improved version)


```
public class MyDialog extends JDialog {
    MyDialog(JFrame parent, String title, boolean modal){
        super(parent, title, modal);
      this.setSize(200, 80);
      this.setLocationRelativeTo(null);
      this.setResizable(false);
        this.initComponent();
      this.setVisible(true);
    }
    public void initComponent() {
        /* initialize components and add them into the
          content pane by calling
          this.getContentPane().add(...);
        */
    }
}
```

Plan

1. Menu
2. Dialog window
3. **Other components: JFileChooser, JColorChooser**

File Choosers

- Provide a GUI for navigating the file system, and then either choosing a file or directory from a list, or entering the name of a file or directory.
- To display a file chooser: use the `JFileChooser` API to show a modal dialog containing the file chooser.
  - Another way: add an instance of JFileChooser to a container.
- The `JFileChooser` API makes it easy to create open and saved dialogs.
- To know more:
  - https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

Example


```
//Create a file chooser
final JFileChooser fc = new JFileChooser();
// ...
//In response to a button click:
int returnVal = fc.showOpenDialog(aComponent);

// ...

if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = fc.getSelectedFile();
    // ...
} else {
    // ...
}
```

Color Choosers

- Enable users to choose from a palette of colors.
- Is a component that you can place anywhere within your program GUI.
- The JColorChooser API also makes it easy to bring up a dialog (modal or not) that contains a color chooser.
- To know more:
  - https://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html

QUESTION ?