Swings Intro

Objectives

After completing this lesson, you should be able to do the following:

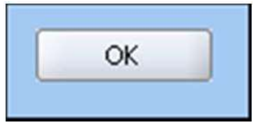
- Explain Abstract Window Toolkit (AWT), Swing, and Java Foundation Classes (JFC)
- Detail the Swing UI containment hierarchy
- Describe how to use layout managers
- Add UI containers to an application to group components
- Embed UI components into UI containers

AWT, Swing, and JFC

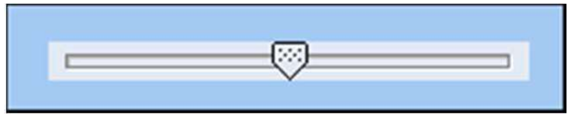
- AWT, or Abstract Window Toolkit (`java.awt`):
  - A graphical user interface library
  - The predecessor to Swing components and the foundation for Swing and JFC
- Swing (`javax.swing`):
  - A more powerful graphical user interface library
  - Built on top of the AWT class hierarchy
- Java Foundation Classes (JFC):
  - A collection of APIs including AWT, Swing, Accessibility API, Pluggable Look and Feel
  - Java 2D API, drag-and-drop support (since JDK 1.2)

Swing Features

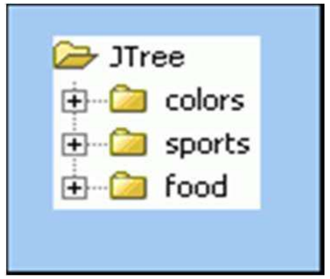
- Swing is a set of visual components that have been available since JDK 1.1 and have been part of the core JDK since version 1.2.
  - Lightweight components compared to AWT
  - Pluggable look-and-feel API
  - Many more components than AWT



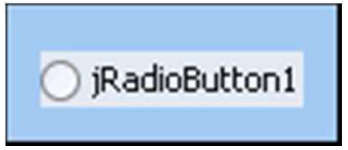
JButton



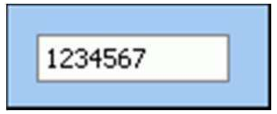
JSlider



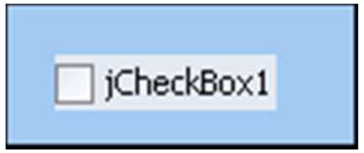
JTree



JRadioButton



JTextField



JCheckBox

Lightweight and Heavyweight Components

Heavyweight components

- Strong dependency on native peer code
- Each rendered in *its own* opaque window
- Early AWT components were mostly heavyweight
- Include some top-level Swing components (`JFrame`, `JApplet`, `JDialog`)

Lightweight components

- No dependence on native peer code
- Can have transparent backgrounds
- Most Swing components are lightweight
- When displayed, can appear nonrectangular
- Must be displayed in heavyweight container

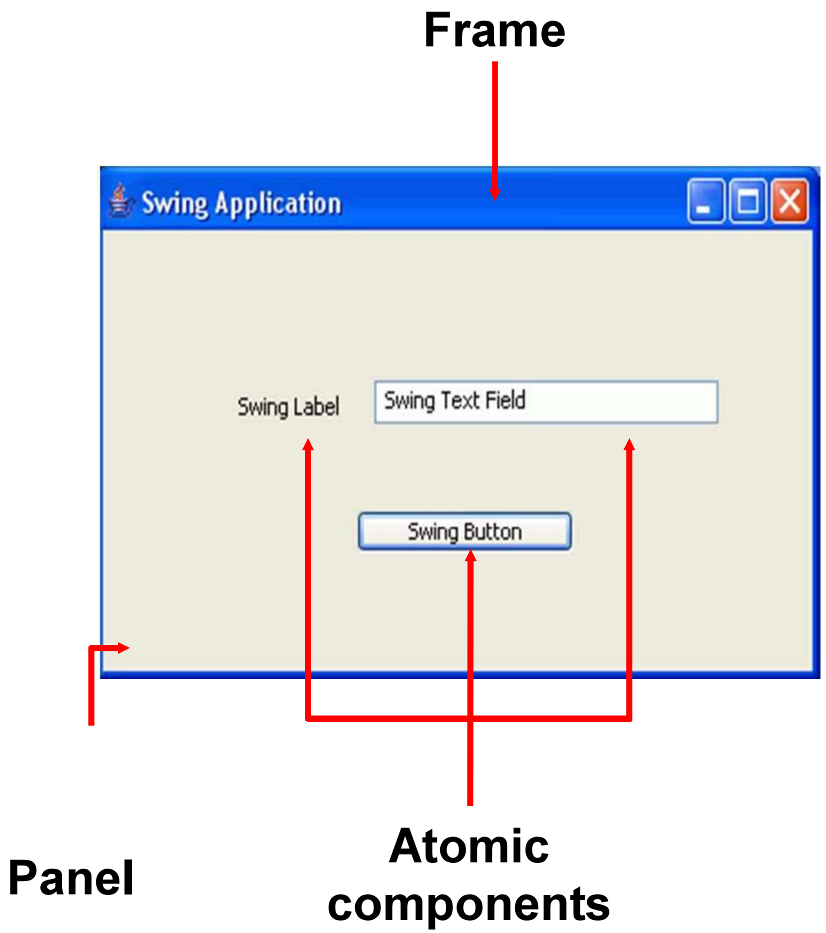
Planning the UI Layout

Building a UI application involves planning, even more so when building Swing applications. Planning requires understanding the following concepts and their relationships:

- UI containment hierarchy (a root component that comprises nested containers and components)
- Container levels and types (such as top-level and intermediate containers)
- Layout managers and their types (used by each container)
- Components that can be added to containers

Swing Containment Hierarchy

- Top-level containers
  - Frame
  - Dialog
  - Applet
- Intermediate containers
  - Panel
  - Scroll Pane
- Atomic components
  - Label
  - Text item
  - Button



Top-Level Containers

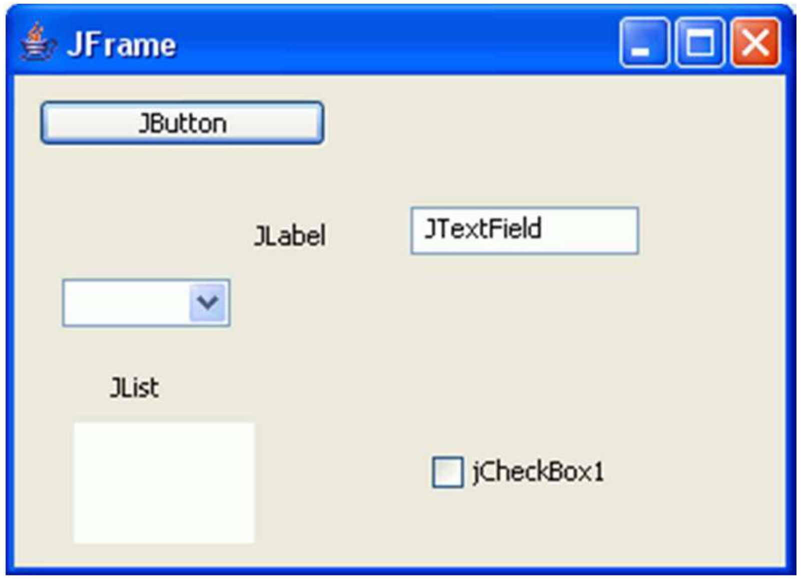
- Swing provides `JFrame`, `JDialog`, and `JApplet`, which have changeable properties such as:
  - Content panes for holding intermediate containers or components, using the `getContentPane()` or `setContentPane()` methods
  - Borders, using a `setBorder()` method
  - Titles, using a `setTitle()` method
  - Window decorations, such as buttons for closing and minimizing (excludes applets)
- AWT provides `Frame`, `Dialog`, and `Applet`
  - These do not provide properties such as a content pane or borders.

Intermediate Containers

- Designed to contain components (or containers); can be nested within other containers
- Types of intermediate containers:
  - Panels for grouping containers or components
  - Scroll panes to add scroll bars around components that can grow, such as a list or a text area
  - Split panes to display two components in a fixed area that is adjustable by the user
  - Tabbed panes for containing multiple components, showing only one at a time based on user selection
  - Toolbars for grouping components, such as buttons
  - Internal frames for nested windows

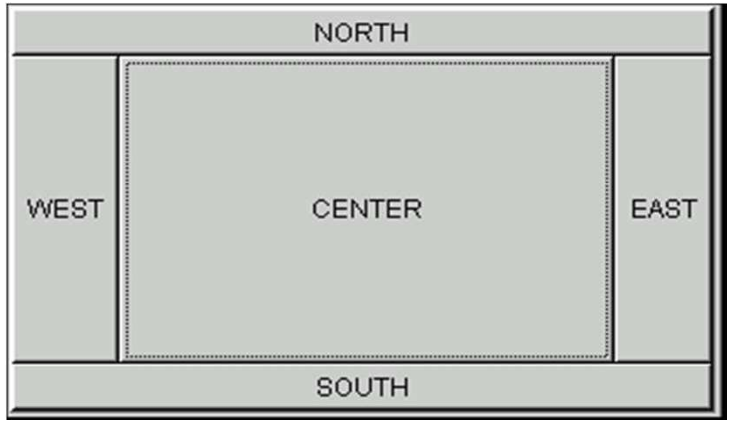
Atomic Components

- Buttons
- Check boxes
- Combo boxes
- Text
- Lists
- Labels

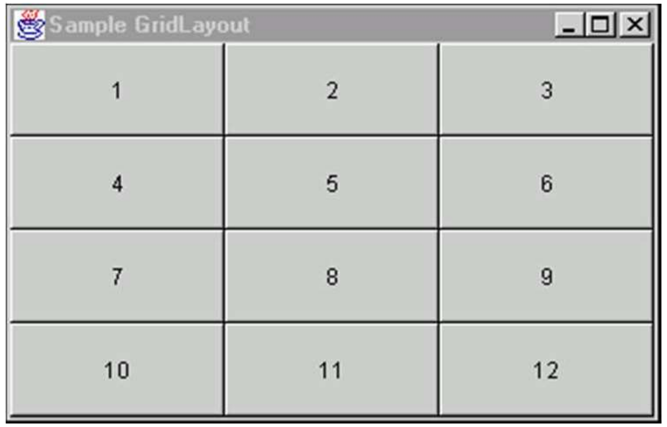


Layout Management: Overview

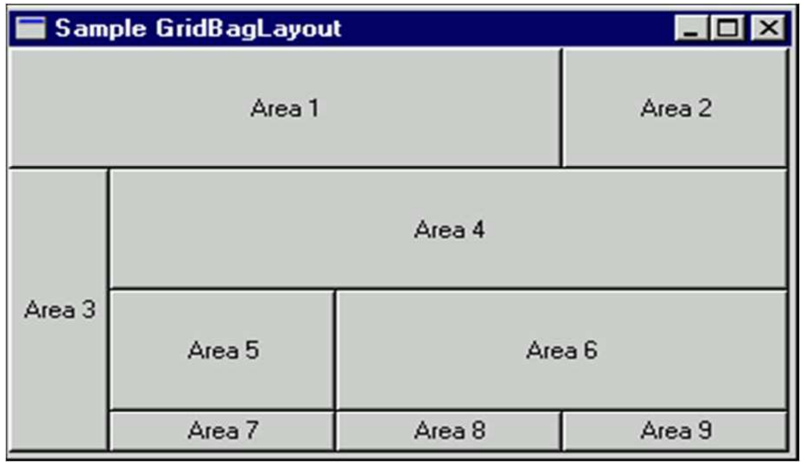
Border layout



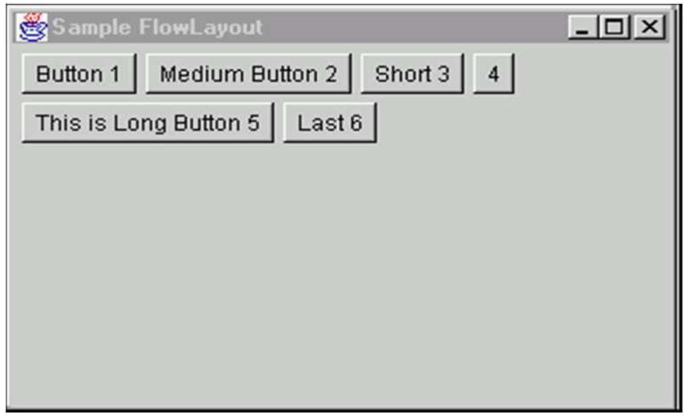
Sample grid layout



Sample gridbag layout

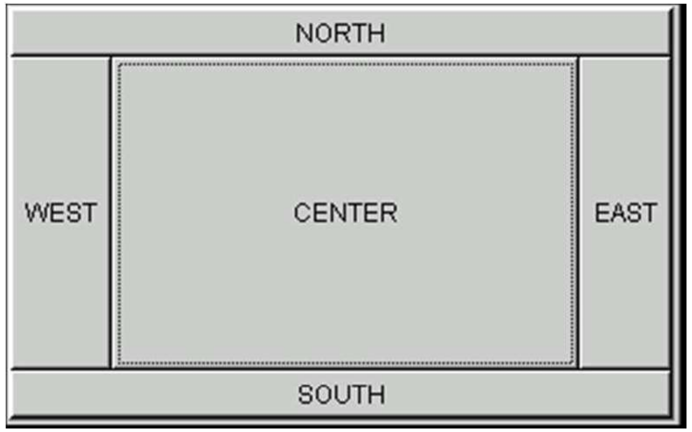


Sample flow layout



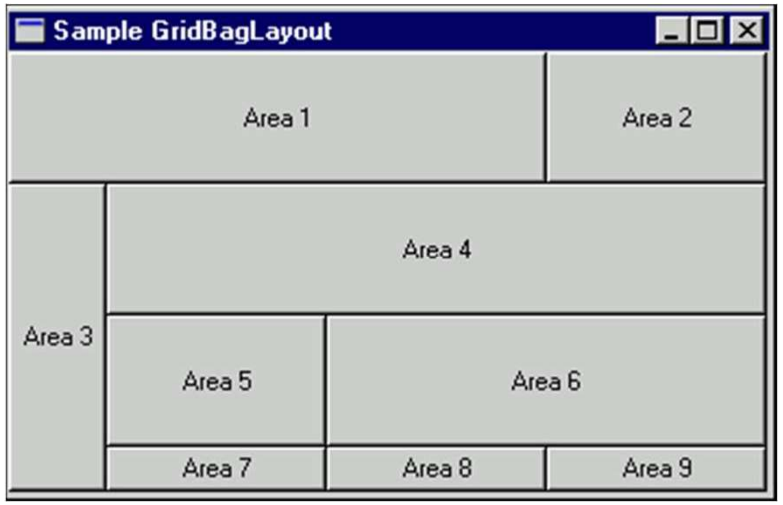
Border Layout

- Has five areas: north, south, east, west, and center
- Has center area that expands to fill the available space
- Displays only one component in each area
- Makes each area useful for holding intermediate panels



GridBag Layout

- Is based on a grid
- Allows components to span multiple rows and columns
- Allows rows and columns to differ in size
- Uses the component’s preferred size to control cell size



GridBag Constraints

External insets

Cell position

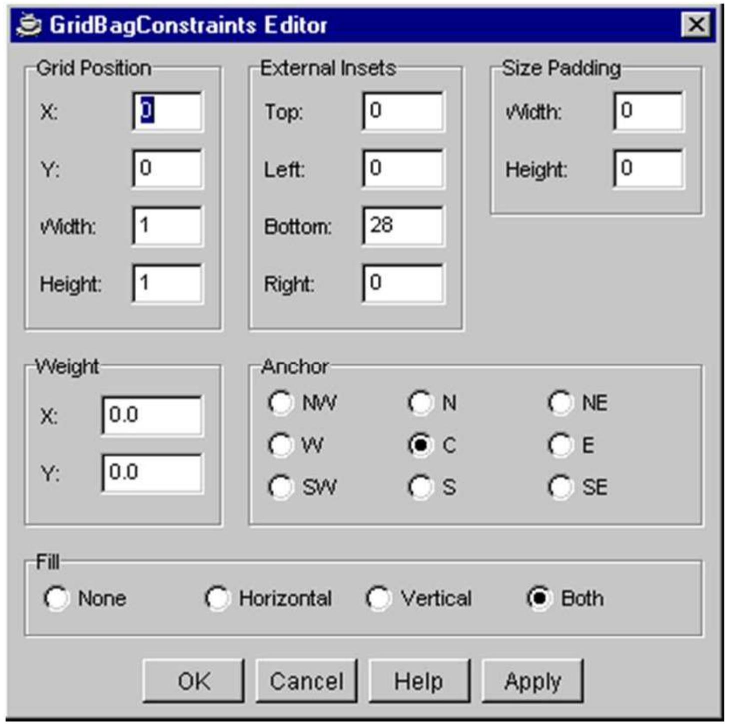
Cell span

Expansion weighting

Fill rules

Component padding

Anchoring



GridBagConstraints Editor

Grid Position
X: 0
Y: 0
Width: 1
Height: 1

External Insets
Top: 0
Left: 0
Bottom: 28
Right: 0

Size Padding
Width: 0
Height: 0

Weight
X: 0.0
Y: 0.0

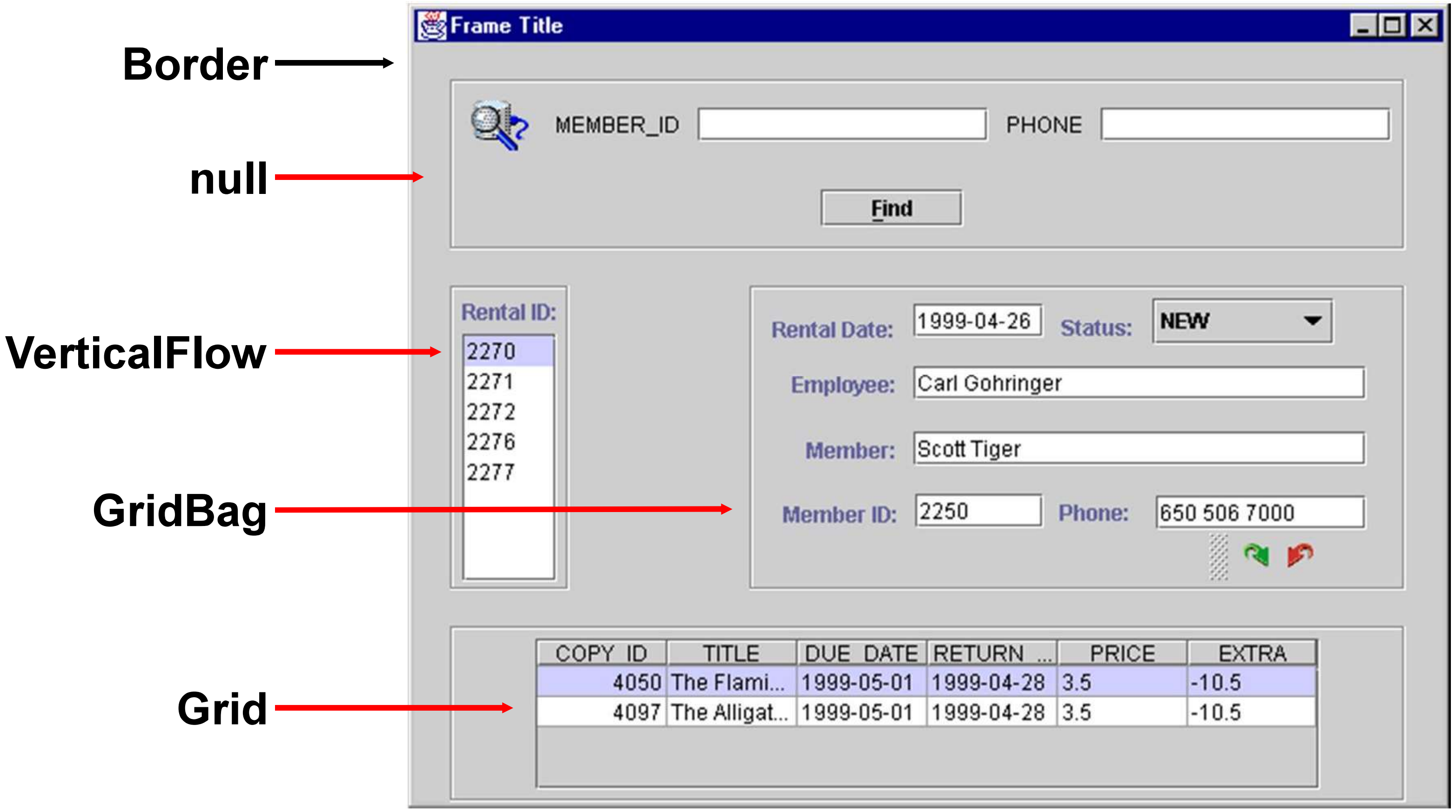
Anchor
○ NW ○ N ○ NE
○ W ● C ○ E
○ SW ○ S ○ SE

Fill
○ None ○ Horizontal ○ Vertical ● Both

Using Layout Managers

- Layout managers are designed to manage multiple components simultaneously.
- Using a layout manager with containers requires:
  - Creating a container and a layout manager object
  - Setting the layout property of the container
  - Adding items (components or other containers) to the regions that are defined by the layout manager
- Different layout managers require different arguments to control component placement.

Combining Layout Managers



Border

null

VerticalFlow

GridBag

Grid

MEMBER_ID

PHONE

Rental ID:

2270
2271
2272
2276
2277

Rental Date: 1999-04-26

Status: NEW

Employee: Carl Gohringer

Member: Scott Tiger

Member ID: 2250

Phone: 650 506 7000

COPY ID	TITLE	DUE DATE	RETURN ...	PRICE	EXTRA
4050	The Flami...	1999-05-01	1999-04-28	3.5	-10.5
4097	The Alligat...	1999-05-01	1999-04-28	3.5	-10.5

Java Frame Classes

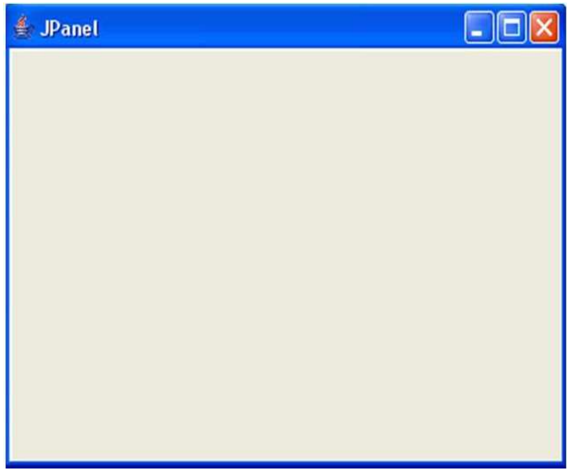
A Java frame is equivalent to an application window.

- Use `JFrame` for a main window.
  - It has properties for icons, title, and the buttons to minimize, maximize, and close.
  - It uses `BorderLayout` by default.
  - It provides a default content pane that occupies the center region of the layout.
  - You can set the frame size with the `setSize()` method and make it visible by using the `setVisible()` method.
- Use `JDialog` for a modal window.
  - You must dismiss a modal window before the application that invokes it can become active.

JPanel Containers

JPanel is a general-purpose container:

- Can use any layout manager (uses Flowlayout by default)
- Can use any border
- Can have added components or other panels or containers by using the add() method



```
JPanel myPanel = new JPanel(new BorderLayout());
JTextArea jTextArea1 = new JTextArea();
myPanel.setBorder(BorderFactory.createRaisedBevelBorder());
myPanel.add(jTextArea1, BorderLayout.SOUTH);
```

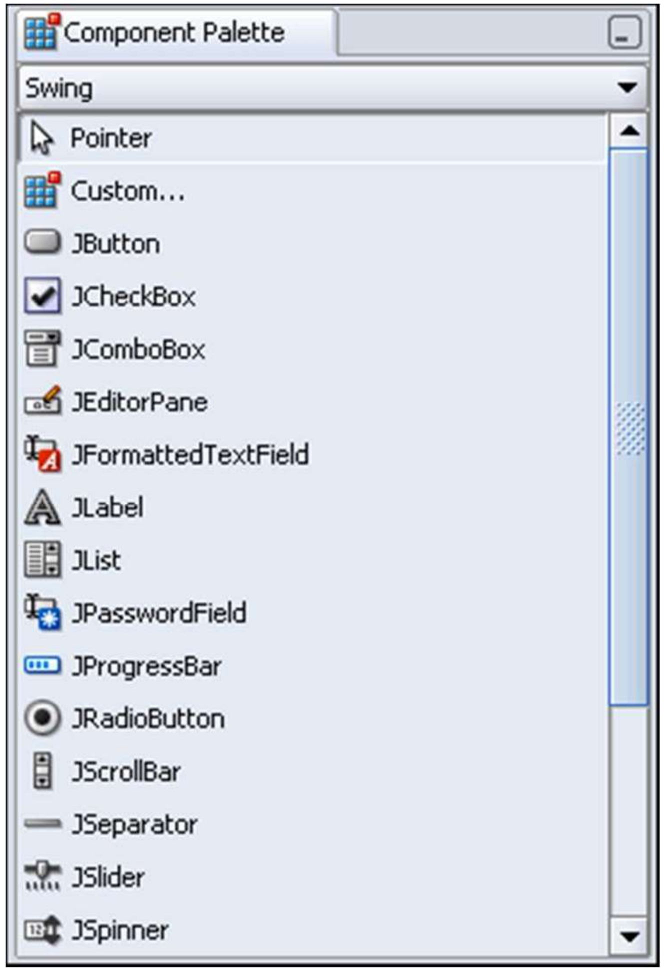
Internal Frames

An internal frame is the equivalent of a document window that is contained in an application window for multiple document interface (MDI) window applications.

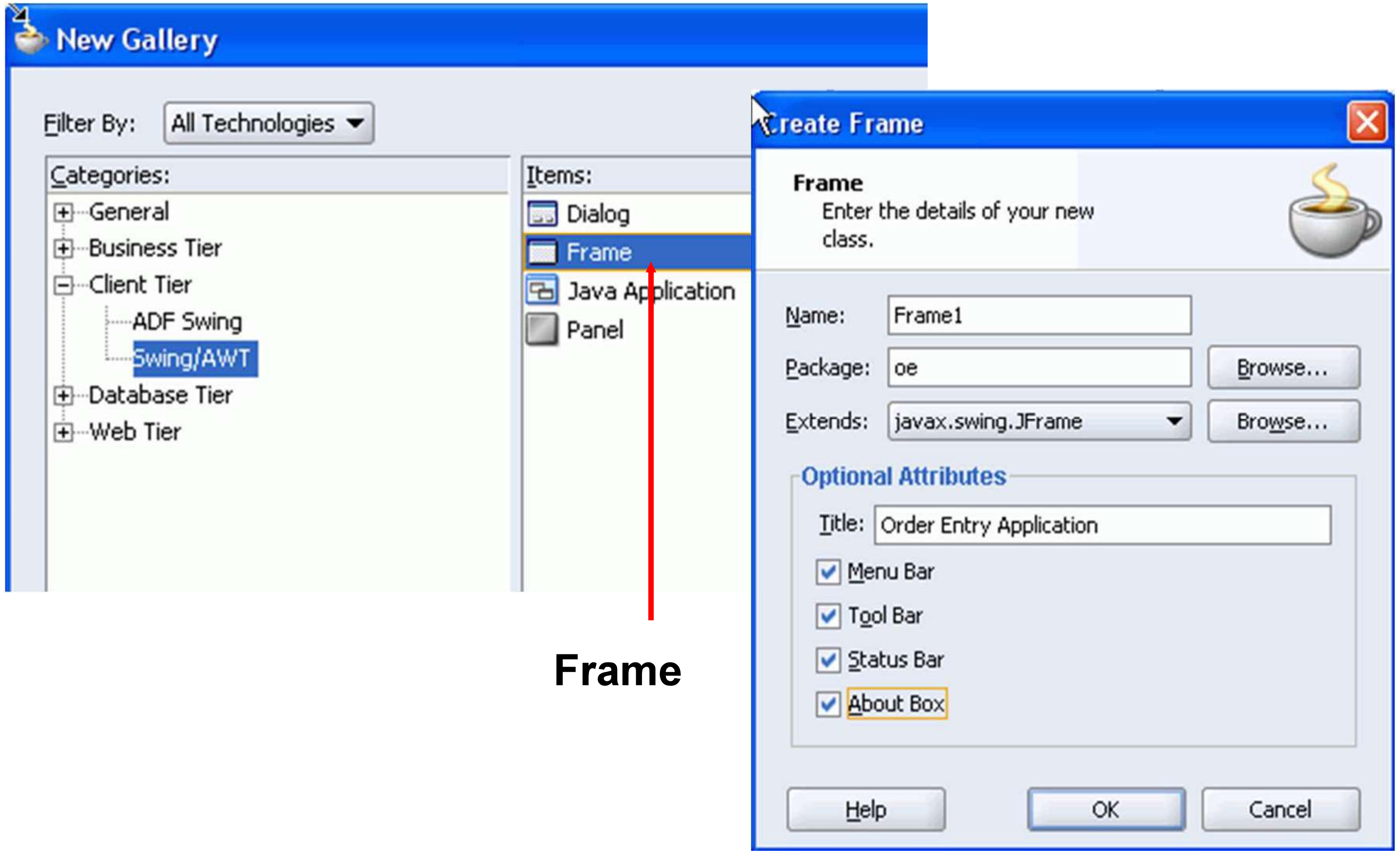
- Use `JInternalFrame` for an internal window:
  - Similar to `JFrame`, it can contain intermediate containers and components and use a layout manager.
  - By default, it is not “closable,” “iconifiable,” “maximizable,” or visible.
- Use `JDesktopPane` as the content pane in which the internal frames are added:
  - Controls the size and placement of internal frames
  - Uses a null layout manager by default

Adding Components

1. Create a `Jframe`.
2. Select a layout manager.
3. Add components from the Component Palette.
4. Fine-tune component properties.



Creating a Frame



New Gallery

Filter By: All Technologies

Categories:
- General
- Business Tier
- Client Tier
  - ADF Swing
  - Swing/AWT
- Database Tier
- Web Tier

Items:
- Dialog
- Frame
- Java Application
- Panel

Create Frame

Frame
Enter the details of your new class.

Name: Frame1

Package: oe

Extends: javax.swing.JFrame

Optional Attributes

Title: Order Entry Application

☑ Menu Bar

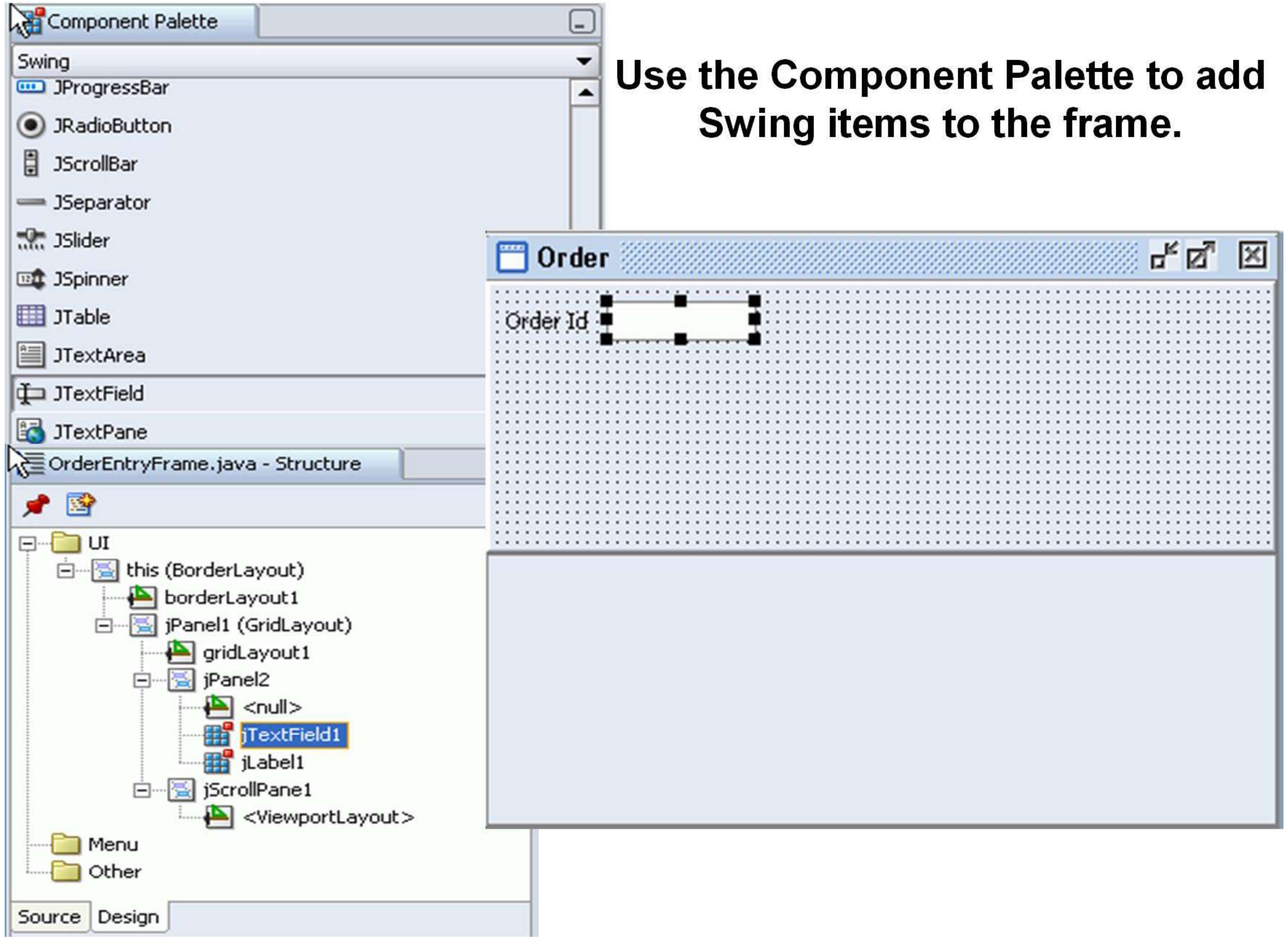
☑ Tool Bar

☑ Status Bar

☑ About Box

Frame

Adding Components



Use the Component Palette to add Swing items to the frame.

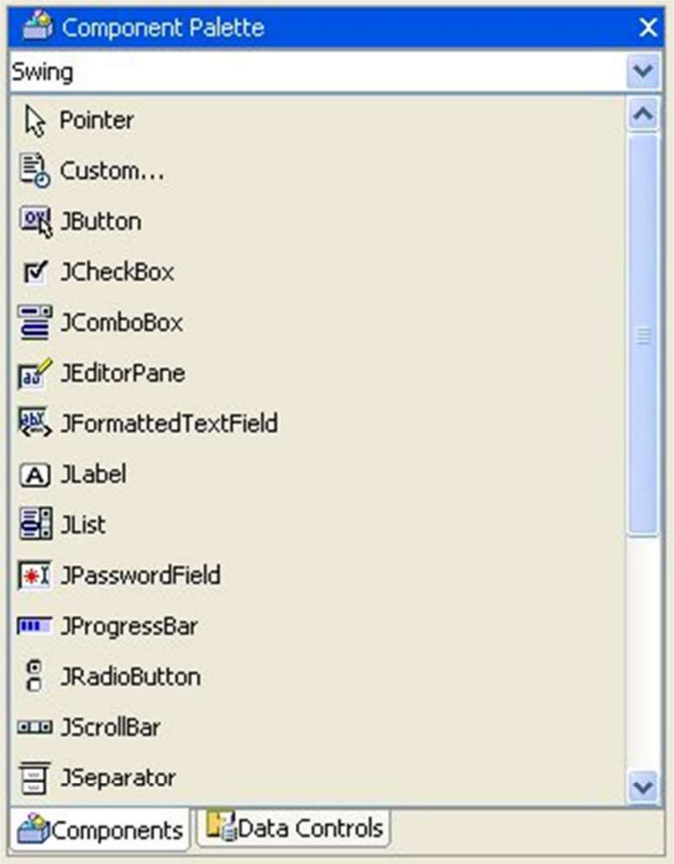
Pluggable Look and Feel

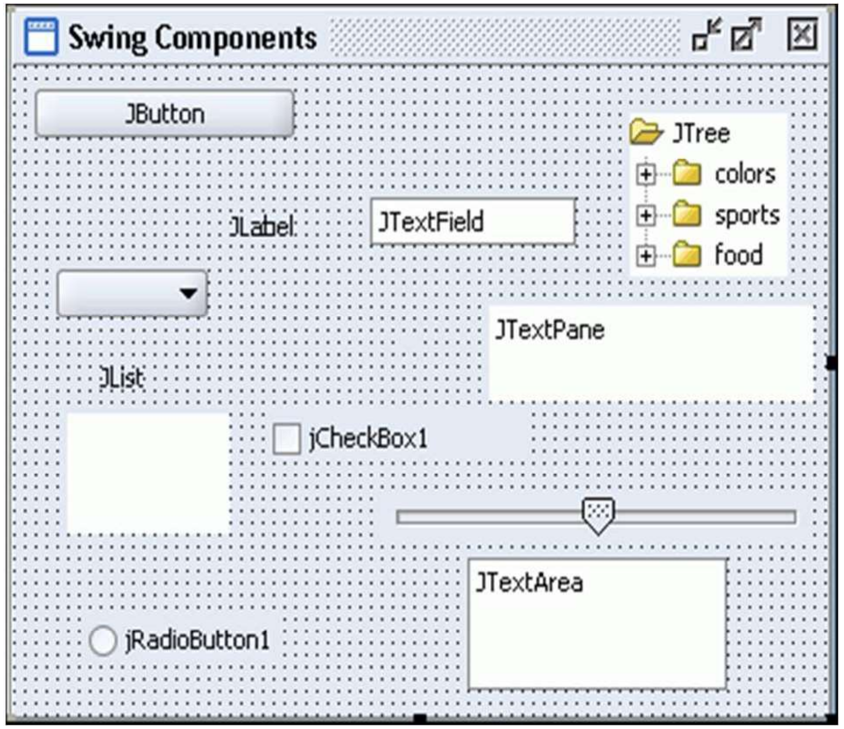
Swing applications provide support for a different look and feel to adapt to the visual environment of the operating system. The look and feel:

- Is application specific:
  - Can be initialized when the application starts
  - Can change dynamically
- Affects lightweight Swing components
- Supports Windows, Macintosh, Java (Metal), and Motif platforms
- Uses the `javax.swing.UIManager` class
  - Provides the `setLookAndFeel()` method, which accepts a look-and-feel class name string.

Swing Components in JDeveloper

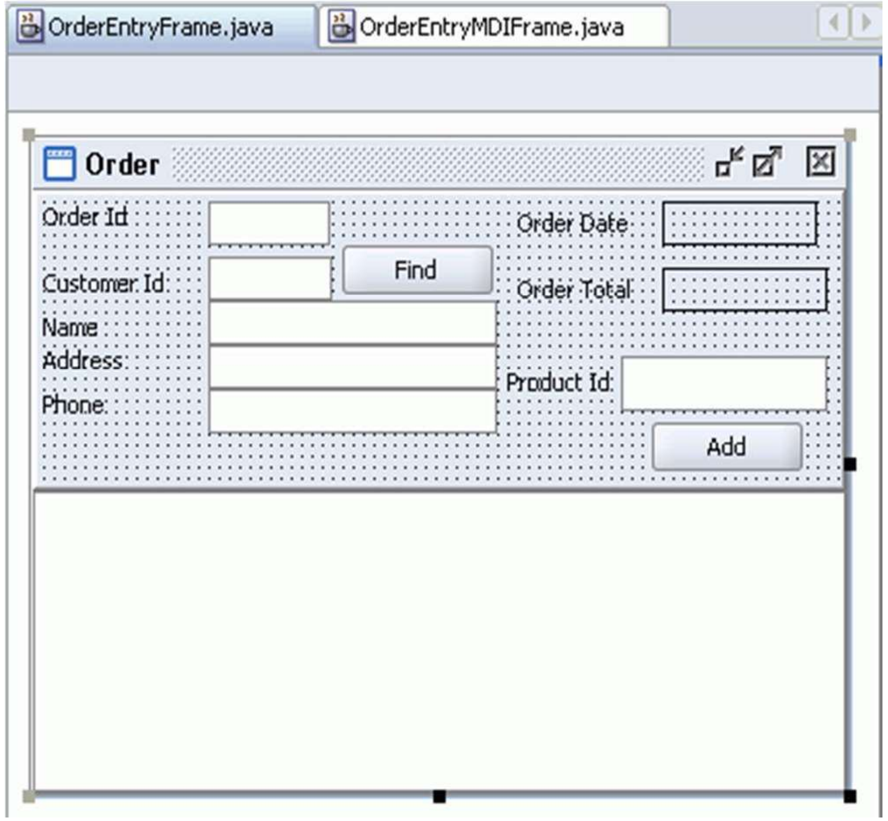
- Use the Swing Component Palette to add items.





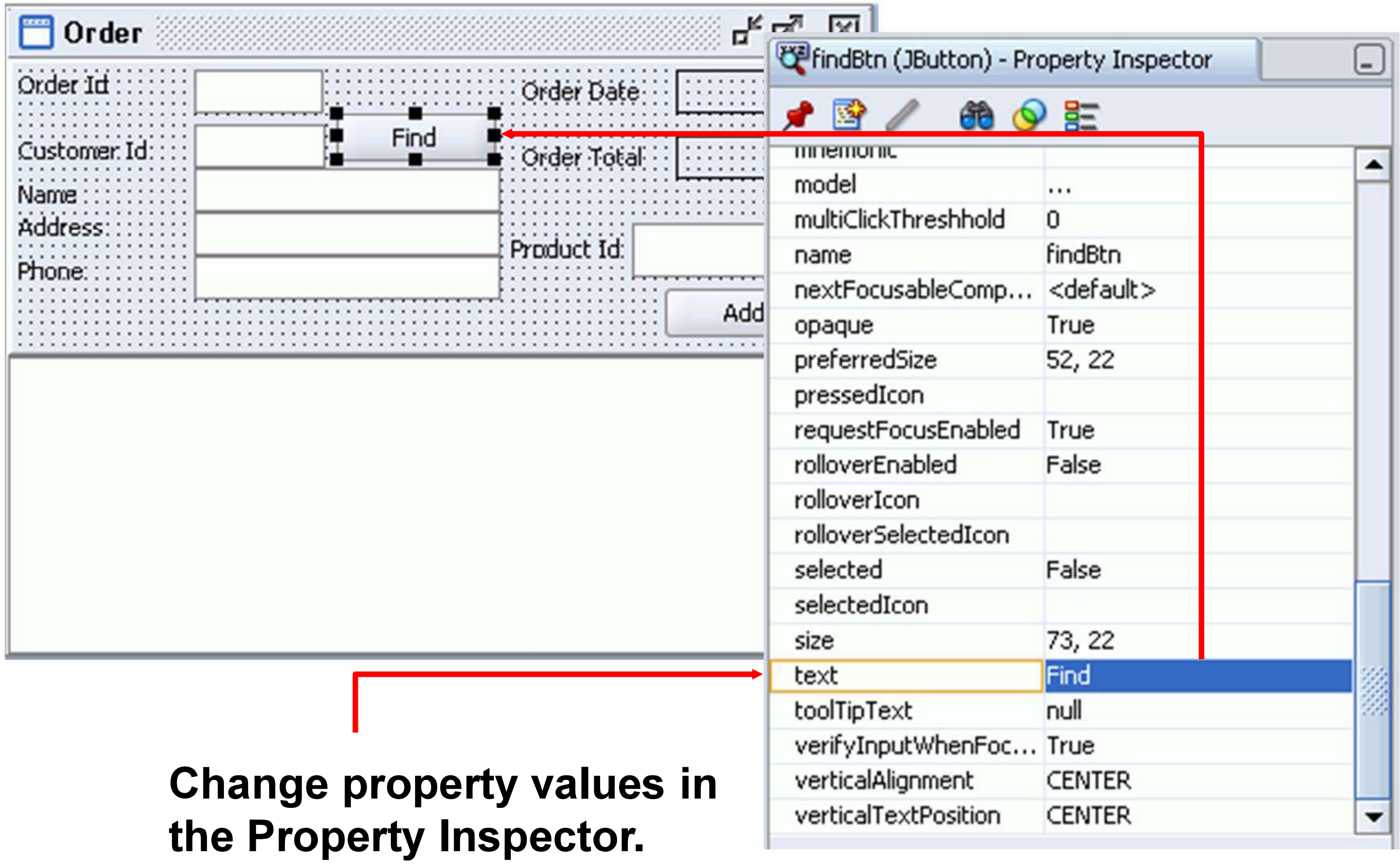
Invoking the UI Editor

Right-click and select Open from the shortcut menu.



UI Editor

Editing the Properties of a Component



findBtn (JButton) - Property Inspector

mnemonic	
model	...
multiClickThreshhold	0
name	findBtn
nextFocusableComp...	<default>
opaque	True
preferredSize	52, 22
pressedIcon	
requestFocusEnabled	True
rolloverEnabled	False
rolloverIcon	
rolloverSelectedIcon	
selected	False
selectedIcon	
size	73, 22
text	Find
toolTipText	null
verifyInputWhenFoc...	True
verticalAlignment	CENTER
verticalTextPosition	CENTER

Change property values in the Property Inspector.

Code Generated by JDeveloper

- Example: Adding `JButton` to `JFrame`

```
import javax.swing.JButton;
public class JFrame1 extends JFrame {
 private JButton jButton1 = new JButton();
 ...
     public void jbInit() throws Exception {
        this.setLayout(null);
        jButton1.setText("jButton1");
        jButton1.setBounds(new Rectangle(25, 140,
                73, 22));
        this.add(jButton1, null);
    }
```

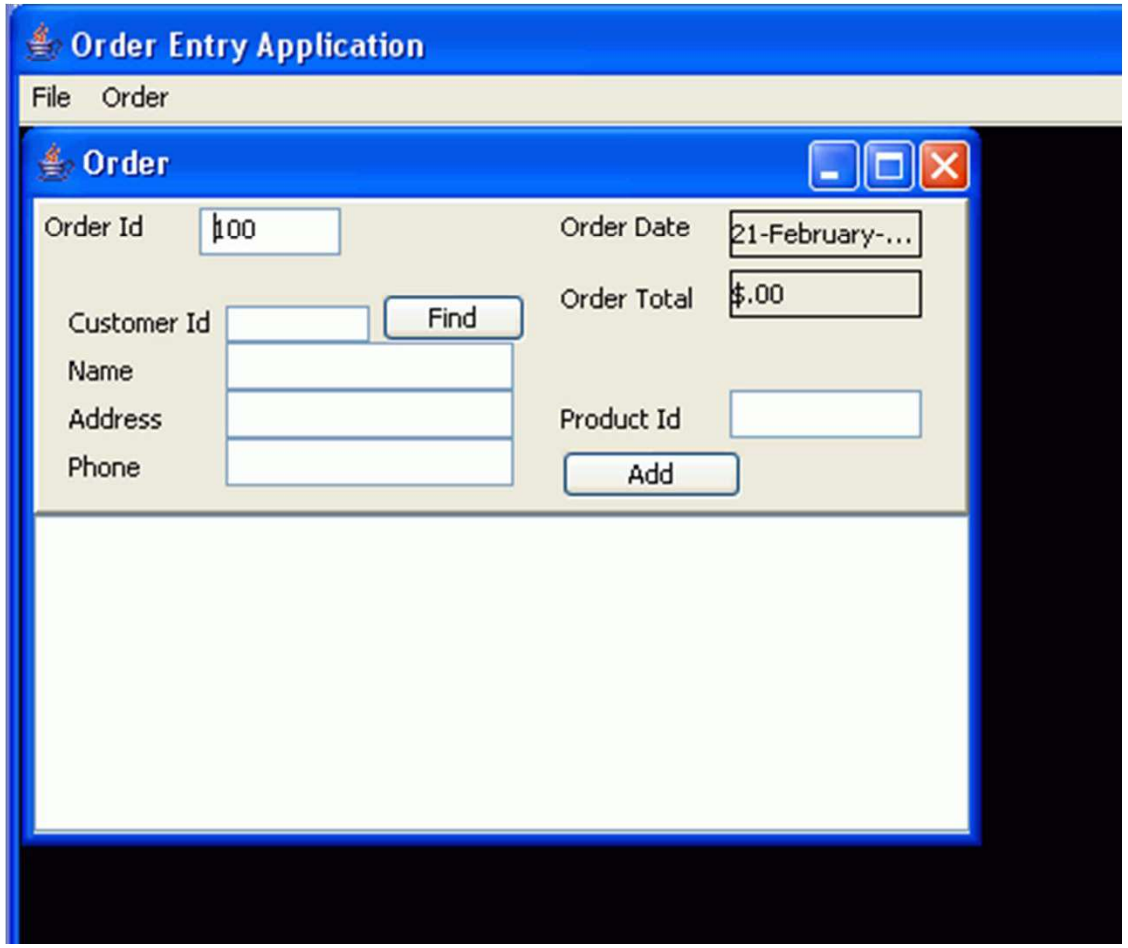
Creating a Menu

- Select Create Menu Bar during application creation.
- Add `JMenuBar` from the Component Palette.
- JDeveloper creates:
  - `JMenuBar` for visual container for menus
  - `JMenu`, which represents a menu of items added to a menu bar
  - `JMenuItems`, which are placed in a `JMenu`
- Each `JMenuItem` supports events, interfaces, and handler methods in the same way as with other Swing UI components.
- `JMenuBar` can be added to any top-level container, such as frames, dialog boxes, and applets.

Practice 17-1 Overview:

- This practice covers the following topics:
  - Creating the `OrderEntryMDIFrame` menu
  - Adding menu items and a separator to the Order menu
  - Adding components to `OrderEntryFrame` to create its visual structure

UI for the Order Entry Application



Order Id 100

Order Date 21-February-...

Order Total $.00

Customer Id

Name

Address

Phone

Product Id

Java Event Handling Model

- How it works:
  - Event originates from source and generates an event object.
  - An event listener hears a specific event.
  - An event handler determines what to do.
- Setting it up:
  1. Create an event source object.
  2. Create an event listener object implementing an interface with methods to handle the event object.
  3. Write an event-specific method to handle the event.
  4. Register the listener object with the event source for the specified event.

Event Handling Code Basics

- Create the event source.

```
Jbutton findBtn = new Jbutton("Find");
```

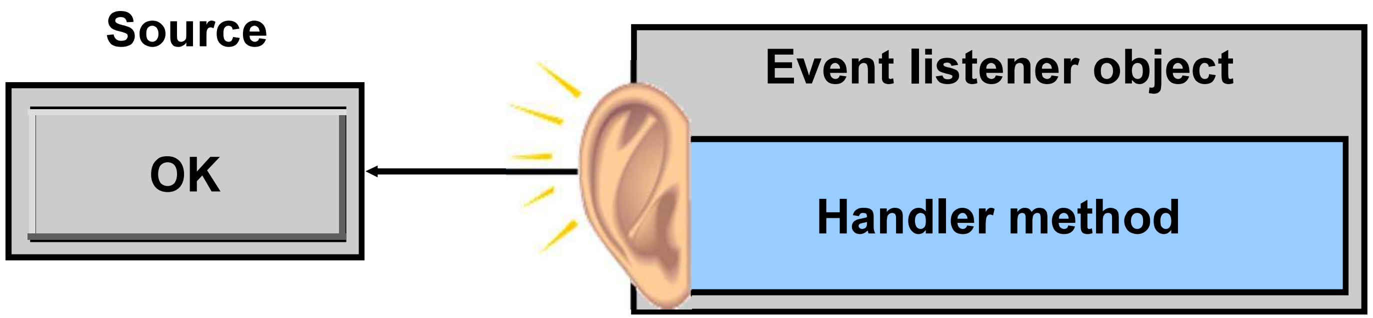
- Create the event listener implementing the required event interface.

```
class MyListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
   // handler logic
 }
}
```

- Register the listener with the event source.

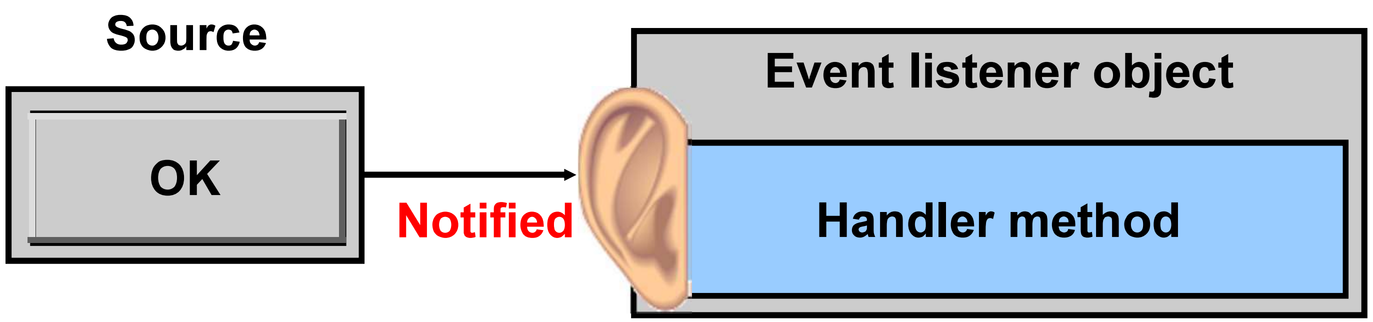
```
findBtn.addActionListener(new MyListener());
```

Event-Handling Process: Registration



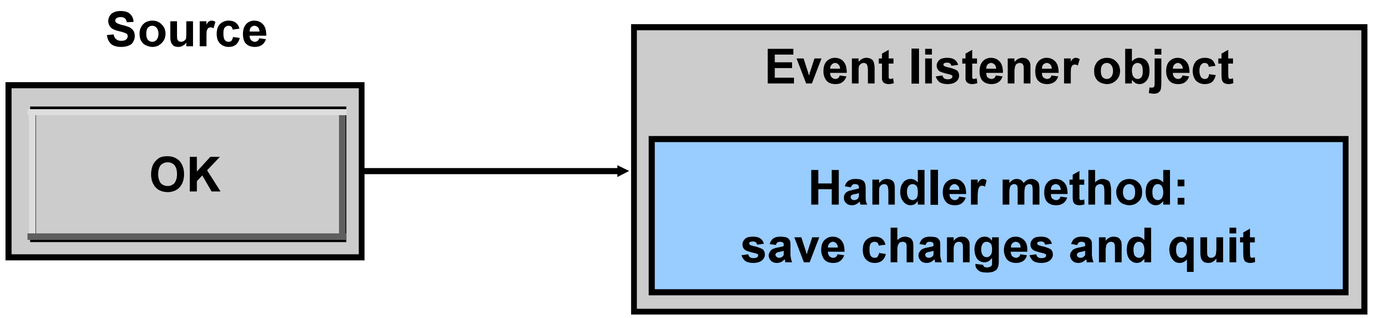
```
MyListener actionListenerObj = new MyListener();
public void jbInit() {
  button1.addActionListener(actionListenerObj);
  …
}
```

Event-Handling Process:



```
public void jbInit() {
  button1.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    // Your code to handle the ActionEvent
   }
  }); … }
```

Event-Handling Process:



Using Adapter Classes for Listeners

Adapter classes are “convenience” classes that implement event listener interfaces:

- They provide empty method implementations.
- They are extended, and the desired method is overridden.

```
interface MouseListener {
 // Declares five methods
}
```

```
class MouseAdapter implements MouseListener {
 // Empty implementations of all five methods
}
```

```
public class MyListener extends MouseAdapter {
 // Override only the methods you need
}
```

Basic Text Component Methods

- Text item (`JLabel`, `JTextField`, and `JButton`) methods:

```
void setText(String value)

String getText()
```

- Additional methods in `JTextArea`:

```
void append(String value)

void insert(String value, int pos)
```

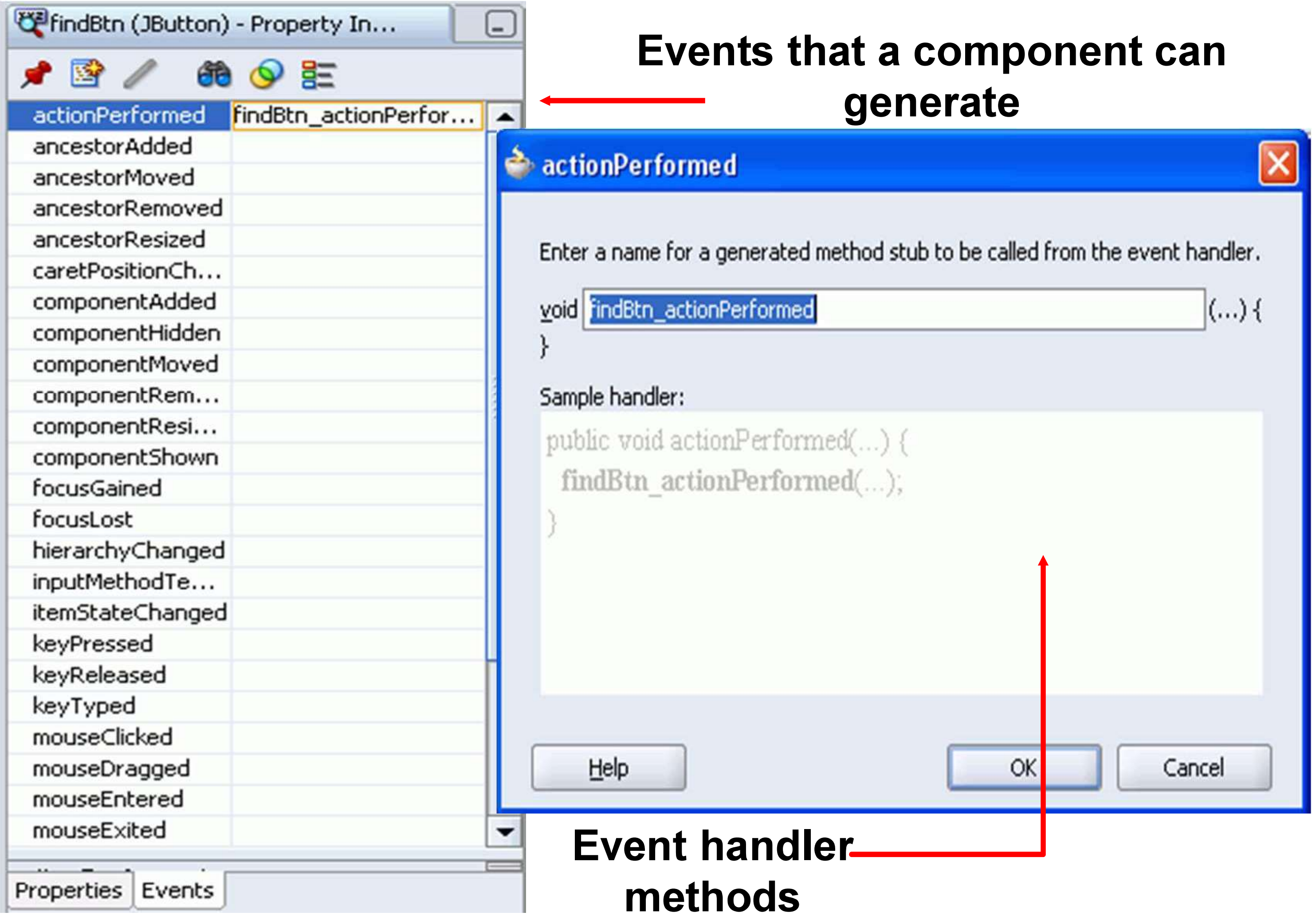
- Changes to component contents are usually made in the event handling thread.

- Note: Consult the Java API documentation for details about each component's capabilities.

Basic `JList` Component Methods

- Subset of `JList` component methods include:
  - `void setListData(Vector)`
    - Copies `Vector` to a `ListModel` applied with `setModel`
  - `void setModel(ListModel)`
    - Sets model representing the data and clears selection
    - Uses the `DefaultListModel` class for the model
  - `Object getSelectedValue()`
    - Returns the selected object, or returns `null` if nothing is selected
  - `int getSelectedIndex()`
    - Returns the index of the selected item, or returns `-1` if nothing is selected

What Events Can a Component Generate?



findBtn (JButton) - Property In...

actionPerformed	findBtn_actionPerfor...
ancestorAdded	
ancestorMoved	
ancestorRemoved	
ancestorResized	
caretPositionCh...	
componentAdded	
componentHidden	
componentMoved	
componentRem...	
componentResi...	
componentShown	
focusGained	
focusLost	
hierarchyChanged	
inputMethodTe...	
itemStateChanged	
keyPressed	
keyReleased	
keyTyped	
mouseClicked	
mouseDragged	
mouseEntered	
mouseExited	

actionPerformed

Enter a name for a generated method stub to be called from the event handler.

```
void findBtn_actionPerformed(...) {
}
```

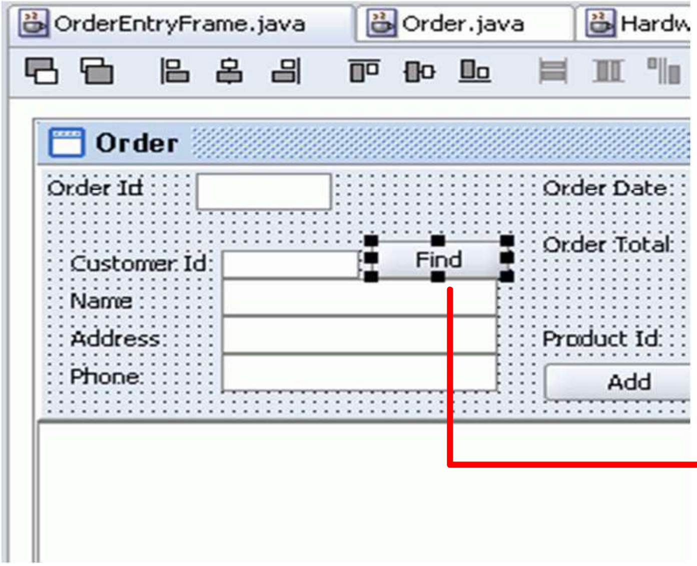
Sample handler:

```
public void actionPerformed(...) {
  findBtn_actionPerformed(...);
}
```

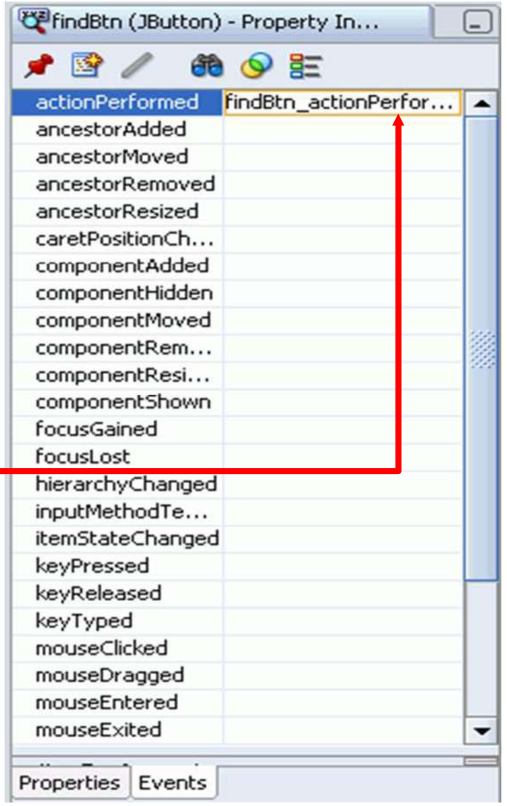
Events that a component can generate

Event handler methods

How to Define an Event Handler



1. Select the event that you want to handle.

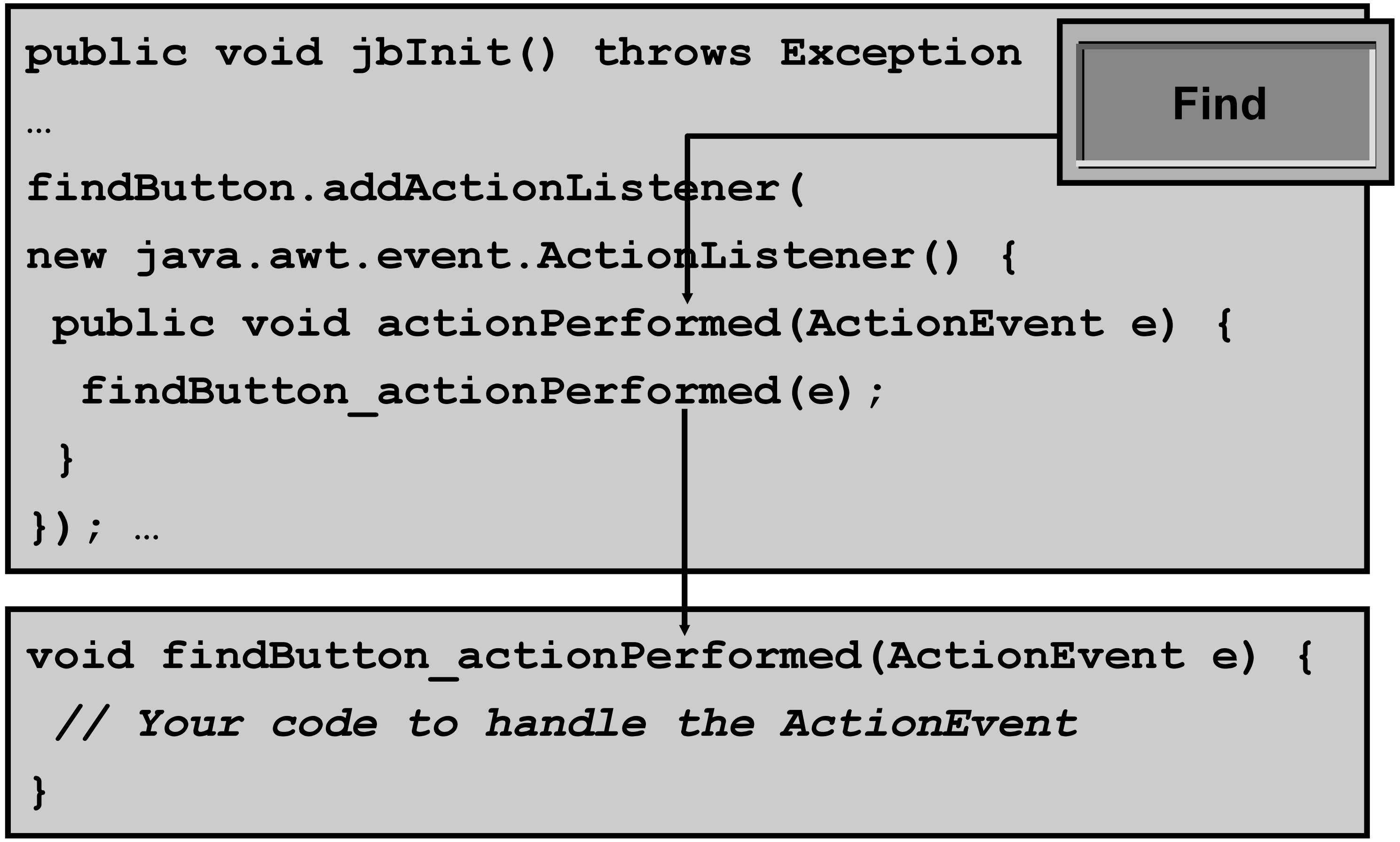


actionPerformed	findBtn_actionPerfor...
ancestorAdded	
ancestorMoved	
ancestorRemoved	
ancestorResized	
caretPositionCh...	
componentAdded	
componentHidden	
componentMoved	
componentRem...	
componentResi...	
componentShown	
focusGained	
focusLost	
hierarchyChanged	
inputMethodTe...	
itemStateChanged	
keyPressed	
keyReleased	
keyTyped	
mouseClicked	
mouseDragged	
mouseEntered	
mouseExited	

2. Click the right column to fill in a method name.

3. Double-click the right column to create the method.

Default Event Handling Code Style Generated by JDeveloper



```
public void jbInit() throws Exception
…
findButton.addActionListener(
new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  findButton_actionPerformed(e);
 }
}); …
```

```
void findButton_actionPerformed(ActionEvent e) {
 // Your code to handle the ActionEvent
}
```

Summary

In this lesson, you should have learned the following:

- Frames are top-level containers.
- Panels are intermediate containers that can be nested.
- Layout managers control component placement.
- Create a menu bar with menus and menu items
- Event Handling