SWING II

CIS*2430 (Fall 2010)

Window Listeners

- Clicking the close-window button on a `JFrame` fires a *window event*
  - Window events are objects of the class `WindowEvent`
- The `setWindowListener` method can register a window listener for a window event
  - A window listener can be programmed to respond to this type of event
  - A window listener is any class that satisfies the `WindowListener` interface

Window Listeners

- A class that implements the `WindowListener` interface must have definitions for all seven method headers in this interface

- Should a method not be needed, it is defined with an empty body

```
public void windowDeiconified(WindowEvent e)
{ }
```

Methods in WindowListener (1/2)

Display 19.1 Methods in the `WindowListener` Interface

The `WindowListener` interface and the `WindowEvent` class are in the package `java.awt.event`.

```
public void windowOpened(WindowEvent e)
```

Invoked when a window has been opened.

```
public void windowClosing(WindowEvent e)
```

Invoked when a window is in the process of being closed. Clicking the close-window button causes an invocation of this method.

```
public void windowClosed(WindowEvent e)
```

Invoked when a window has been closed.

(continued)

Methods in WindowListener (2/2)

Display 19.1 Methods in the `WindowListener` Interface

```
public void windowIconified(WindowEvent e)
```

Invoked when a window is iconified. When you click the minimize button in a `JFrame`, it is iconified.

```
public void windowDeiconified(WindowEvent e)
```

Invoked when a window is deiconified. When you activate a minimized window, it is deiconified.

```
public void windowActivated(WindowEvent e)
```

Invoked when a window is activated. When you click in a window, it becomes the activated window. Other actions can also activate a window.

```
public void windowDeactivated(WindowEvent e)
```

Invoked when a window is deactivated. When a window is activated, all other windows are deactivated. Other actions can also deactivate a window.

A Window Listener (1/8)

Display 19.2 A Window Listener

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
```

(continued)

A Window Listener (2/8)

Display 19.2 A Window Listener

```
public class WindowListenerDemo extends JFrame
{
    public static final int WIDTH = 300; //for main window
    public static final int HEIGHT = 200; //for main window
    public static final int SMALL_WIDTH = 200; //for confirm window
    public static final int SMALL_HEIGHT = 100;//for confirm window

    private class CheckOnExit implements WindowListener
    {
        public void windowOpened(WindowEvent e)
        {}

        public void windowClosing(WindowEvent e)
        {
            ConfirmWindow checkers = new ConfirmWindow();
            checkers.setVisible(true);
        }
```

This `WindowListener` *class is an inner class.*

(continued)

A Window Listener (3/8)

Display 19.2 A Window Listener

```
public void windowClosed(WindowEvent e)
{}

public void windowIconified(WindowEvent e)
{}

 public void windowDeiconified(WindowEvent e)
{}

public void windowActivated(WindowEvent e)
{}
```

A window listener must define all the method headings in the `WindowListener` *interface, even if some are trivial implementations.*

(continued)

A Window Listener (4/8)

Display 19.2 A Window Listener

```
        public void windowDeactivated(WindowEvent e)
        {}
    } //End of inner class CheckOnExit

    private class ConfirmWindow extends JFrame implements ActionListener
    {
        public ConfirmWindow()
        {
            setSize(SMALL_WIDTH, SMALL_HEIGHT);
            getContentPane().setBackground(Color.YELLOW);
            setLayout(new BorderLayout());

            JLabel confirmLabel = new JLabel(
                            "Are you sure you want to exit?");
            add(confirmLabel, BorderLayout.CENTER);
```

Another inner class.

(continued)

A Window Listener (5/8)

Display 19.2 A Window Listener

```
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.ORANGE);
        buttonPanel.setLayout(new FlowLayout());

        JButton exitButton = new JButton("Yes");
        exitButton.addActionListener(this);
        buttonPanel.add(exitButton);

        JButton cancelButton = new JButton("No");
        cancelButton.addActionListener(this);
        buttonPanel.add(cancelButton);

        add(buttonPanel, BorderLayout.SOUTH);
    }
```

(continued)

A Window Listener (6/8)

Display 19.2 A Window Listener

```
        public void actionPerformed(ActionEvent e)
        {
            String actionCommand = e.getActionCommand();

            if (actionCommand.equals("Yes"))
                System.exit(0);
            else if (actionCommand.equals("No"))
                dispose();//Destroys only the ConfirmWindow.
            else
                System.out.println("Unexpected Error in Confirm Window.");
        }
    } //End of inner class ConfirmWindow
```

(continued)

A Window Listener (7/8)

Display 19.2 A Window Listener

```
    public static void main(String[] args)
    {
        WindowListenerDemo demoWindow = new WindowListenerDemo();
        demoWindow.setVisible(true);
    }

    public WindowListenerDemo()
    {
        setSize(WIDTH, HEIGHT);
        setTitle("Window Listener Demonstration");

        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new CheckOnExit());

        getContentPane().setBackground(Color.LIGHT_GRAY);
        JLabel aLabel = new JLabel("I like to be sure you are sincere.");
        add(aLabel);
    }
}
```

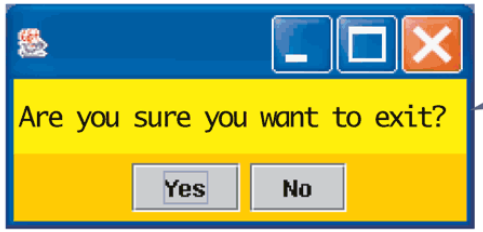
Even if you have a window listener, you normally must still invoke `setDefaultCloseOperation`.

(continued)

A Window Listener (8/8)

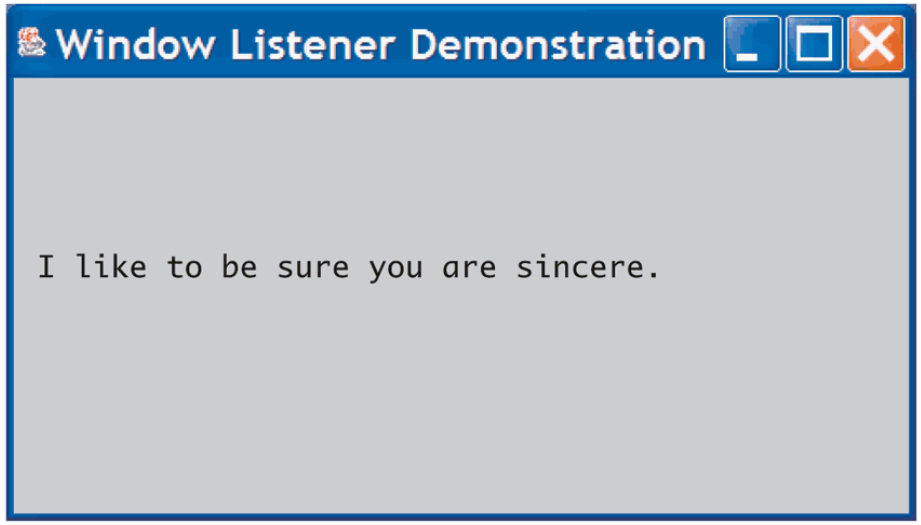
Display 19.2 A Window Listener

RESULTING GUI



This window is an object of the class `ConfirmWindow`.

When you click this close-window button, the second window appears.



The `dispose` Method

- The **`dispose`** method of the **`JFrame`** class eliminates the invoking **`JFrame`** without ending the program
  - The resources consumed by this **`JFrame`** and its components are returned for reuse
  - Unless all the elements are eliminated (i.e., in a one window program), this does not end the program

- **`dispose`** is often used in a program with multiple windows to eliminate one window without ending the program

Pitfall: setDefaultCloseOperation

- The behavior of `setDefaultCloseOperation` takes place even if there is a window listener registered to the `JFrame`
  - Whether or not a window listener is registered to respond to window events, a `setDefaultCloseOperation` invocation should be included
  - This invocation is usually made in the `JFrame` constructor

Pitfall: setDefaultCloseOperation

- If the window listener takes care of all of the window behavior, then the `JFrame` constructor should contain the following:

```
setDefaultCloseOperation(
                JFrame.DO_NOTHING_ON_CLOSE)
```

- If it is not included, the following default action will take place instead, regardless of whether or not a window listener is supposed to take care of it:

```
setDefaultCloseOperation(
                JFrame.HIDE_ON_CLOSE);
```

The `WindowAdapter` Class

- When a class does not give true implementations to most of the method headings in the `WindowListener` interface, it may be better to make it a derived class of the `WindowAdapter` class
  - Only the method headings used need be defined
  - The other method headings inherit trivial implementation from `WindowAdapter`, so there is no need for empty method bodies

The WindowAdapter Class

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
…

public class WindowListenerDemo extends JFrame {
  …
  private class CheckOnExit extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
      ConfirmWindow checkers = new ConfirmWindow();
      checkers.setVisible(true);
    }
  }
  …
}
```

Listeners as Inner Classes

- Often, instead of having one action listener object deal with all the action events in a GUI, a separate `ActionListener` class is created for each button or menu item
  - Each button or menu item has its own unique action listener
  - There is then no need for a multiway if-else statement
- When this approach is used, each class is usually made a private inner class

Listeners as Inner Classes (1/6)

Display 17.16 Listeners as Inner Classes

<Import statements are the same as in Display 17.14.>

```
public class InnerListenersDemo extends JFrame
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    private JPanel redPanel;
    private JPanel whitePanel;
    private JPanel bluePanel;
```

(continued)

Listeners as Inner Classes (2/6)

Display 17.16 Listeners as Inner Classes

```
private class RedListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        redPanel.setBackground(Color.RED);
    }
} //End of RedListener inner class
```

```
private class WhiteListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        whitePanel.setBackground(Color.WHITE);
    }
} //End of WhiteListener inner class
```

(continued)

Listeners as Inner Classes (3/6)

Display 17.16 Listeners as Inner Classes

```
22      private class BlueListener implements ActionListener
23      {
24          public void actionPerformed(ActionEvent e)
25          {
26              bluePanel.setBackground(Color.BLUE);
27          }
28      } //End of BlueListener inner class

29      public static void main(String[] args)
30      {
31          InnerListenersDemo gui = new InnerListenersDemo();
32          gui.setVisible(true);
33      }
```

(continued)

Listeners as Inner Classes (4/6)

Display 17.16 Listeners as Inner Classes

```
    public InnerListenersDemo()
    {
        super("Menu Demonstration");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(1, 3));

        redPanel = new JPanel();
        redPanel.setBackground(Color.LIGHT_GRAY);
        add(redPanel);

        whitePanel = new JPanel();
        whitePanel.setBackground(Color.LIGHT_GRAY);
        add(whitePanel);
```

The resulting GUI is the same as in Display 17.14.

(continued)

Listeners as Inner Classes (5/6)

Display 17.16 Listeners as Inner Classes

```
bluePanel = new JPanel();
bluePanel.setBackground(Color.LIGHT_GRAY);
add(bluePanel);

JMenu colorMenu = new JMenu("Add Colors");

JMenuItem redChoice = new JMenuItem("Red");
redChoice.addActionListener(new RedListener());
colorMenu.add(redChoice);
```

(continued)

Listeners as Inner Classes (6/6)

Display 17.16 Listeners as Inner Classes

```
        JMenuItem whiteChoice = new JMenuItem("White");
        whiteChoice.addActionListener(new WhiteListener());
        colorMenu.add(whiteChoice);

        JMenuItem blueChoice = new JMenuItem("Blue");
        blueChoice.addActionListener(new BlueListener());
        colorMenu.add(blueChoice);

        JMenuBar bar = new JMenuBar();
        bar.add(colorMenu);
        setJMenuBar(bar);
    }

}
```

Icons

- `JLabels`, `JButtons`, and `JMenuItems` can have icons
  - An *icon* is just a small picture (usually)
  - It is not required to be small
- An icon is an object of the `ImageIcon` class
  - It is based on a digital picture file such as `.gif`, `.jpg`, or `.tiff`
- Labels, buttons, and menu items may display a string, an icon, a string and an icon, or nothing

Icons

- An icon can be added to a label using the `setIcon` method as follows:

```
JLabel dukeLabel = new JLabel("Mood
  check");
ImageIcon dukeIcon = new
  ImageIcon("duke_waving.gif");
dukeLabel.setIcon(dukeIcon);
```

- Instead, an icon can be given as an argument to the `JLabel` constructor:

```
JLabel dukeLabel = new JLabel(dukeIcon);
```

- Text can be added to the label as well using the `setText` method:

```
dukeLabel.setText("Mood check");
```

Icons

- Icons and text may be added to **JButtons** and **JMenuItems** in the same way as they are added to a **JLabel**

```
JButton happyButton = new
  JButton("Happy");
ImageIcon happyIcon = new
                ImageIcon("smiley.gif");
happyButton.setIcon(happyIcon);
```

Using Icons (1/5)

Display 19.4 Using Icons

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.ImageIcon;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;


public class IconDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 500;
    public static final int HEIGHT = 200;
    public static final int TEXT_FIELD_SIZE = 30;

    private JTextField message;
```

(continued)

Using Icons (2/5)

Display 19.4 Using Icons

```
    public static void main(String[] args)
    {
        IconDemo iconGui = new IconDemo();
        iconGui.setVisible(true);
    }


    public IconDemo()
    {
        super("Icon Demonstration");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setBackground(Color.WHITE);
        setLayout(new BorderLayout());
```

(continued)

Using Icons (3/5)

Display 19.4 Using Icons

```
JLabel dukeLabel = new JLabel("Mood check");
ImageIcon dukeIcon = new ImageIcon("duke_waving.gif");
dukeLabel.setIcon(dukeIcon);
add(dukeLabel, BorderLayout.NORTH);

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout());
JButton happyButton = new JButton("Happy");
ImageIcon happyIcon = new ImageIcon("smiley.gif");
happyButton.setIcon(happyIcon);
happyButton.addActionListener(this);
buttonPanel.add(happyButton);
JButton sadButton = new JButton("Sad");
ImageIcon sadIcon = new ImageIcon("sad.gif");
sadButton.setIcon(sadIcon);
```

(continued)

Using Icons (4/5)

Display 19.4 Using Icons

```
        sadButton.addActionListener(this);
        buttonPanel.add(sadButton);
        add(buttonPanel, BorderLayout.SOUTH);

        message = new JTextField(TEXT_FIELD_SIZE);
        add(message, BorderLayout.CENTER);
    }


    public void actionPerformed(ActionEvent e)
    {
       String actionCommand = e.getActionCommand();
```

(continued)

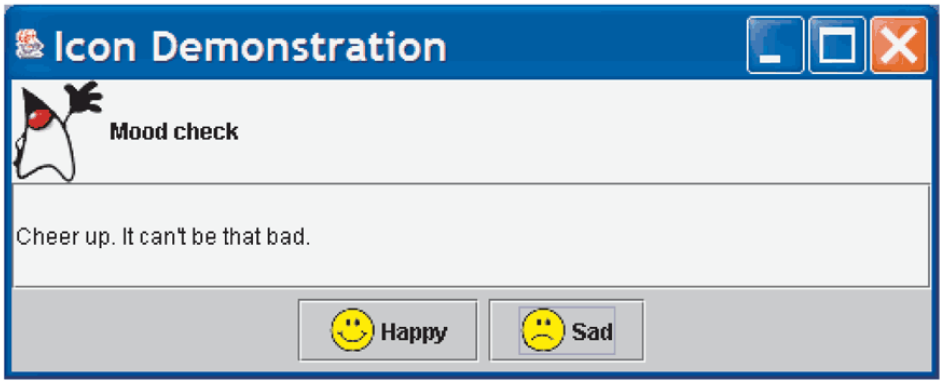
Using Icons (5/5)

Display 19.4 Using Icons

```
        if (actionCommand.equals("Happy"))
            message.setText(
                    "Smile and the world smiles with you!");
        else if (actionCommand.equals("Sad"))
            message.setText(
                   "Cheer up. It can't be that bad.");
        else
             message.setText("Unexpected Error.");
    }
}
```

RESULTING GUI'

View after clicking the "Sad" button.



Changing Visibility

- A GUI can have components that change from visible to invisible and back again

- In the following example, the label with the character *Duke* not waving is shown first
  - When the "**`Wave`**" button is clicked, the label with *Duke* not waving disappears and the label with *Duke* waving appears
  - When the "**`Stop`**" button is clicked, the label with *Duke* waving disappears, and the label with *Duke* not waving returns
  - *Duke* is Sun Microsystem's mascot for the Java Language

Changing Visibility (1/6)

Display 19.9 Labels with Changing Visibility

```
import javax.swing.JFrame;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class VisibilityDemo extends JFrame
                            implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
```

(continued)

Changing Visibility (2/6)

Display 19.9 Labels with Changing Visibility

```
    private JLabel wavingLabel;
    private JLabel standingLabel;
    public static void main(String[] args)
    {
        VisibilityDemo demoGui = new VisibilityDemo();
        demoGui.setVisible(true);
    }

    public VisibilityDemo()
    {
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Visibility Demonstration");

        setLayout(new BorderLayout());
```

(continued)

Changing Visibility (3/6)

Display 19.9 Labels with Changing Visibility

```
JPanel picturePanel = new JPanel();
picturePanel.setBackground(Color.WHITE);
picturePanel.setLayout(new FlowLayout());

ImageIcon dukeStandingIcon =
          new ImageIcon("duke_standing.gif");
standingLabel = new JLabel(dukeStandingIcon);
standingLabel.setVisible(true);
picturePanel.add(standingLabel);

ImageIcon dukeWavingIcon = new ImageIcon("duke_waving.gif");
wavingLabel = new JLabel(dukeWavingIcon);
wavingLabel.setVisible(false);
picturePanel.add(wavingLabel);
```

(continued)

Changing Visibility (4/6)

Display 19.9 Labels with Changing Visibility

```
        add(picturePanel, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.LIGHT_GRAY);
        buttonPanel.setLayout(new FlowLayout());

        JButton waveButton = new JButton("Wave");
        waveButton.addActionListener(this);
        buttonPanel.add(waveButton);

        JButton stopButton = new JButton("Stop");
        stopButton.addActionListener(this);
        buttonPanel.add(stopButton);
```

(continued)

Changing Visibility (5/6)

Display 19.9 Labels with Changing Visibility

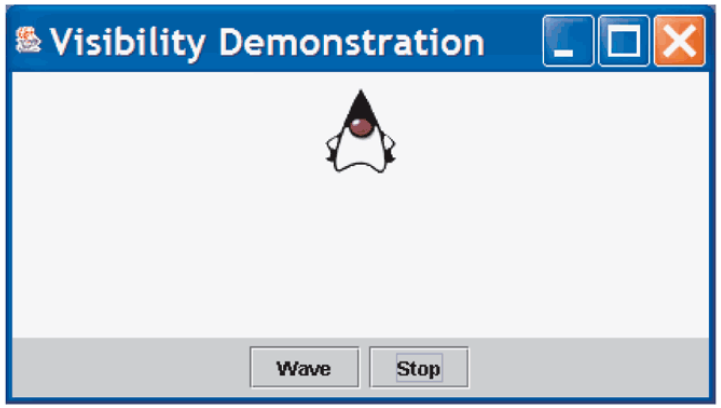
```
        add(buttonPanel, BorderLayout.SOUTH);
    }
    public void actionPerformed(ActionEvent e)
    {
        String actionCommand = e.getActionCommand();

        if (actionCommand.equals("Wave"))
        {
            wavingLabel.setVisible(true);
            standingLabel.setVisible(false);
        }
        else if (actionCommand.equals("Stop"))
        {
            standingLabel.setVisible(true);
            wavingLabel.setVisible(false);
        }
        else
            System.out.println("Unanticipated error.");
    }
}
```

(continued)

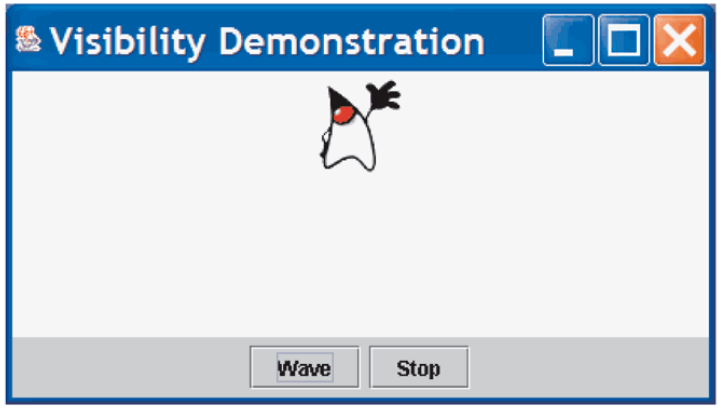
Changing Visibility (6/6)

Display 19.9 Labels with Changing Visibility

RESULTING GUI (After clicking Stop button)



RESULTING GUI (After clicking Wave button)



Dimension Class

- Objects of the **Dimension** class can be used with buttons, menu items, and other objects to specify a size
  - The **Dimension** class is in the package **java.awt**
    Dimension(int width, int height)
  - Example:

```
JButton aButton = new JButton("Enter");
aButton.setPreferredSize(
            new Dimension(30, 50));
```

The `Insets` Class

- Objects of the class **`Insets`** can be used to specify the size of the margin in a button or menu item
  - The **`Insets`** class is in the package **`java.awt`**

```
public Insets(int top, int left,
              int bottom, int right)
```

  - Example:

```
aButton.setMargin(
        new Insets(10,20,10,20));
```

`getActionCommand` Method

- When a user clicks a button or menu item, an event is fired that normally goes to one or more action listeners
  - The action event becomes an argument to an **`actionPerformed`** method
  - This action event includes a **`String`** instance variable called the *action command* for the button or menu item
  - This string can be retrieved with **`getActionCommand`** method
    `e.getActionCommand()`

setActionCommand Method

- The **setActionCommand** method can be used to change the action command for a component
  - This is especially useful when two or more buttons or menu items have the same default action command strings

```
JButton nextButton = new JButton("Next");
nextButton.setActionCommand("Next Button");

JMenuItem choose = new JMenuItem("Next");
choose.setActionCommand("Next Menu Item");
```

Methods in AbstractButton (1/3)

Display 17.15 Some Methods in the Class AbstractButton

The abstract class AbstractButton is in the javax.swing package.
All of these methods are inherited by both of the classes JButton and JMenuItem.

```
public void setBackground(Color theColor)
```

Sets the background color of this component.

```
public void addActionListener(ActionListener listener)
```

Adds an ActionListener.

```
public void removeActionListener(ActionListener listener)
```

Removes an ActionListener.

```
public void setActionCommand(String actionCommand)
```

Sets the action command.

(continued)

Methods in AbstractButton (2/3)

Display 17.15 Some Methods in the Class AbstractButton

```
public String getActionCommand()
```

Returns the action command for this component.

```
public void setText(String text)
```

Makes text the only text on this component.

```
public String getText()
```

Returns the text written on the component, such as the text on a button or the string for a menu item.

```
public void setPreferredSize(Dimension preferredSize)
```

Sets the preferred size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to use the preferred size. The following special case will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setPreferredSize(
                        new Dimension(int width, int height))
```

(continued)

Methods in AbstractButton (3/3)

Display 17.15 Some Methods in the Class AbstractButton

```
public void setMaximumSize(Dimension maximumSize)
```

Sets the maximum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this maximum size. The following special case will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setMaximumSize(
                        new Dimension(int width, int height))
```

```
public void setMinimumSize(Dimension minimumSize)
```

Sets the minimum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this minimum size.
Although we do not discuss the `Dimension` class, the following special case is intuitively clear and will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setMinimumSize(
                       new Dimension(int width, int height))
```

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.BorderLayout;

public class WindowExample extends JFrame implements ActionListener {
   public static final int WIDTH = 300;
   public static final int DEPTH = 200;

   private JLabel label = null;
   private JButton button = null;

   public static void main(String[] args) {
     WindowExample gui = new WindowExample();
     gui.setVisible(true);
   }
```

```
public WindowExample() {
   super( "Window Example" );
   setSize( WIDTH, DEPTH );
   setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   setLayout( new BorderLayout() );

   JMenu commands = new JMenu( "Commands" );
   JMenuItem close = new JMenuItem( "close" );
   close.addActionListener( this );
   commands.add( close );
   JMenuBar bar = new JMenuBar();
   bar.add( commands );
   setJMenuBar( bar );

   label = new JLabel( "Welcome to Window Example!" );
   add( label, BorderLayout.CENTER );

   button = new JButton( "Close the window" );
   button.addActionListener( this );
 }
```

```
  public void actionPerformed( ActionEvent e ) {
    String command = e.getActionCommand();
    if( command.equals("close") ) {
      remove( label );
      add( button, BorderLayout.SOUTH );
      validate();
    } else if( command.equals("Close the window") )
      System.exit(0);
    else
      System.out.println( "Unexpected error" );
  }

}
```