Chapter 18

Swing II

FIFTH EDITION

ABSOLUTE JAVA

WALTER SAVITCH

Slides prepared by Rose Williams,
Binghamton University

Kenrick Mock, *University of Alaska Anchorage*

PEARSON   ALWAYS LEARNING

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

18-2

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)
{ }
```

18-3

Methods in the **WindowListener** Interface
(Part 1 of 2)

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)

18-4

Methods in the `WindowListener` Interface (Part 2 of 2)

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.

18-5

A Window Listener Inner Class

- An inner class often serves as a window listener for a `JFrame`
  - The following example uses a window listener inner class named `CheckOnExit`
  `addWindowListener(new CheckOnExit());`
- When the close-window button of the main window is clicked, it fires a window event
  - This is received by the anonymous window listener object
- This causes the `windowClosing` method to be invoked

18-6

A Window Listener Inner Class

- The method `windowClosing` creates and displays a `ConfirmWindow` class object
  - It contains the message `"Are you sure you want to exit?"` as well as `"Yes"` and `"No"` buttons
- If the user clicks `"Yes,"` the action event fired is received by the `actionPerformed` method
  - It ends the program with a call to `System.exit`
- If the user clicks `"No,"` the `actionPerformed` method invokes the dispose method
  - This makes the calling object go away (i.e., the small window of the `ConfirmWindow` class), but does not affect the main window

18-7

A Window Listener (Part 1 of 8)

A Window Listener

```
1   import javax.swing.JFrame;
2   import javax.swing.JPanel;
3   import java.awt.BorderLayout;
4   import java.awt.FlowLayout;
5   import java.awt.Color;
6   import javax.swing.JLabel;
7   import javax.swing.JButton;
8   import java.awt.event.ActionListener;
9   import java.awt.event.ActionEvent;
10  import java.awt.event.WindowListener;
11  import java.awt.event.WindowEvent;
```

(continued)

18-8

A Window Listener (Part 2 of 8)

A Window Listener

```
12   public class WindowListenerDemo extends JFrame
13   {
14       public static final int WIDTH = 300; //for main window
15       public static final int HEIGHT = 200; //for main window
16       public static final int SMALL_WIDTH = 200; //for confirm window
17       public static final int SMALL_HEIGHT = 100;//for confirm window

18       private class CheckOnExit implements WindowListener
19       {
20           public void windowOpened(WindowEvent e)
21           {}

22           public void windowClosing(WindowEvent e)
23           {
24               ConfirmWindow checkers = new ConfirmWindow();
25               checkers.setVisible(true);
26           }
```

This WindowListener class is an inner class.

(continued)

18-9

A Window Listener (Part 3 of 8)

A Window Listener

```
27       public void windowClosed(WindowEvent e)
28       {}

29       public void windowIconified(WindowEvent e)
30       {}

31        public void windowDeiconified(WindowEvent e)
32       {}

33       public void windowActivated(WindowEvent e)
34       {}
```

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

(continued)

18-10

A Window Listener (Part 4 of 8)

A Window Listener

```
35           public void windowDeactivated(WindowEvent e)
36           {}
37   } //End of inner class CheckOnExit

38   private class ConfirmWindow extends JFrame implements ActionListener
39   {
40       public ConfirmWindow()
41       {
42           setSize(SMALL_WIDTH, SMALL_HEIGHT);
43           getContentPane().setBackground(Color.YELLOW);
44           setLayout(new BorderLayout());

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

Another inner class.

(continued)

18-11

A Window Listener (Part 5 of 8)

A Window Listener

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

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

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

57           add(buttonPanel, BorderLayout.SOUTH);
58       }
```

(continued)

18-12

A Window Listener (Part 6 of 8)

A Window Listener

```
59          public void actionPerformed(ActionEvent e)
60          {
61              String actionCommand = e.getActionCommand();
62
63              if (actionCommand.equals("Yes"))
64                  System.exit(0);
65              else if (actionCommand.equals("No"))
66                  dispose();//Destroys only the ConfirmWindow.
67              else
68                  System.out.println("Unexpected Error in Confirm Window.");
69          }
       } //End of inner class ConfirmWindow
```

(continued)

18-13

A Window Listener (Part 7 of 8)

A Window Listener

```
70
71      public static void main(String[] args)
72      {
73          WindowListenerDemo demoWindow = new WindowListenerDemo();
74          demoWindow.setVisible(true);
75      }
76
77      public WindowListenerDemo()
78      {
79          setSize(WIDTH, HEIGHT);
80          setTitle("Window Listener Demonstration");
81
82          setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
83          addWindowListener(new CheckOnExit());
84
85          getContentPane().setBackground(Color.LIGHT_GRAY);
86          JLabel aLabel = new JLabel("I like to be sure you are sincere.");
87          add(aLabel);
88      }
89  }
```

Even if you have a window listener, you normally must still invoke setDefaultCloseOperation.
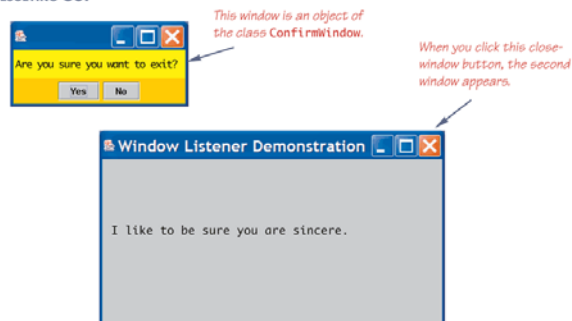
(continued)

18-14

A Window Listener (Part 8 of 8)

A Window Listener

RESULTING GUI

This window is an object of the class ConfirmWindow.

Are you sure you want to exit?
Yes   No

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

Window Listener Demonstration

I like to be sure you are sincere.

18-15

The **dispose** Method

- The **dispose** method of the **JFrame** class is used to eliminate 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

18-16

Pitfall: Forgetting to Invoke `setDefaultCloseOperation`

- The behavior set by the `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

18-17

Pitfall: Forgetting to Invoke `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);
  ```
  - This will hide the `JFrame`, but not end the program

18-18

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
- This can only be done when the `JFrame` does not need to be derived from any other class

18-19

Using `WindowAdapter`

Using WindowAdapter

This requires the following import statements:

```
        import java.awt.event.WindowAdapter;
        import java.awt.event.WindowEvent;
1       private class CheckOnExit extends WindowAdapter
2       {
3           public void windowClosing(WindowEvent e)
4           {
5               ConfirmWindow checkers = new ConfirmWindow();
6               checkers.setVisible(true);
7           }
8       } //End of inner class CheckOnExit
```

18-20

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

18-21

Icons

- The class **ImageIcon** is used to convert a picture file to a Swing icon

  ```
  ImageIcon dukeIcon = new
      ImageIcon("duke_waving.gif");
  ```

  - The picture file must be in the same directory as the class in which this code appears, unless a complete or relative path name is given
  - Note that the name of the picture file is given as a string

18-22

Icons

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

  ```
  JLabel dukeLabel = new JLabel("Mood check");
  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");
  ```

18-23

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);
  ```

18-24

Icons

- Button or menu items can be created with just an icon by giving the **ImageIcon** object as an argument to the **JButton** or **JMenuItem** constructor

```
ImageIcon happyIcon = new
                    ImageIcon("smiley.gif");
JButton smileButton = new JButton(happyIcon);
JMenuItem happyChoice = new
                    JMenuItem(happyIcon);
```

  – A button or menu item created without text should use the **setActionCommand** method to explicitly set the action command, since there is no string

18-25

Using Icons (Part 1 of 5)

Using Icons

```
 1  import javax.swing.JFrame;
 2  import javax.swing.JPanel;
 3  import javax.swing.JTextField;
 4  import javax.swing.ImageIcon;
 5  import java.awt.BorderLayout;
 6  import java.awt.FlowLayout;
 7  import java.awt.Color;
 8  import javax.swing.JLabel;
 9  import javax.swing.JButton;
10  import java.awt.event.ActionListener;
11  import java.awt.event.ActionEvent;

12  public class IconDemo extends JFrame implements ActionListener
13  {
14      public static final int WIDTH = 500;
15      public static final int HEIGHT = 200;
16      public static final int TEXT_FIELD_SIZE = 30;

17      private JTextField message;
```

(continued)

18-26

Using Icons (Part 2 of 5)

Using Icons

```
18      public static void main(String[] args)
19      {
20          IconDemo iconGui = new IconDemo();
21          iconGui.setVisible(true);
22      }


23      public IconDemo()
24      {
25          super("Icon Demonstration");
26          setSize(WIDTH, HEIGHT);
27          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

(continued)

18-27

Using Icons (Part 3 of 5)

Using Icons

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

34          JPanel buttonPanel = new JPanel();
35          buttonPanel.setLayout(new FlowLayout());
36          JButton happyButton = new JButton("Happy");
37          ImageIcon happyIcon = new ImageIcon("smiley.gif");
38          happyButton.setIcon(happyIcon);
39          happyButton.addActionListener(this);
40          buttonPanel.add(happyButton);
41          JButton sadButton = new JButton("Sad");
42          ImageIcon sadIcon = new ImageIcon("sad.gif");
43          sadButton.setIcon(sadIcon);
```

(continued)

18-28

Using Icons (Part 4 of 5)

Using Icons

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

47          message = new JTextField(TEXT_FIELD_SIZE);
48          add(message, BorderLayout.CENTER);
49      }

50      public void actionPerformed(ActionEvent e)
51      {
52          String actionCommand = e.getActionCommand();
```
(continued)

Using Icons   (Part 5 of 5)

Using Icons

```
53          if (actionCommand.equals("Happy"))
54              message.setText(
55                  "Smile and the world smiles with you!");
56          else if (actionCommand.equals("Sad"))
57              message.setText(
58                  "Cheer up. It can't be that bad.");
59          else
60              message.setText("Unexpected Error.");
61      }
62  }
```

RESULTING GUI¹

View after clicking the "Sad" button.
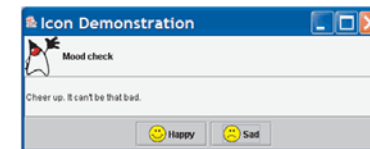
Some Methods in the Classes **JButton**, **JMenuItem**, and **JLabel** (Part 1 of 4)

Some Methods in the Classes JButton, JMenuItem, **and** JLabel

```
public JButton()
public JMenuItem()
public JLabel()
```
Creates a button, menu item, or label with no text or icon on it. (Typically, you will later use setText and/or setIcon with the button, menu item, or label.)

```
public JButton(String text)
public JMenuItem(String text)
public JLabel(String text)
```
Creates a button, menu item, or label with the text on it.

```
public JButton(ImageIcon picture)
public JMenuItem(ImageIcon picture)
public JLabel(ImageIcon picture)
```
Creates a button, menu item, or label with the icon picture on it and no text.

(continued)

Some Methods in the Classes **JButton**, **JMenuItem**, and **JLabel** (Part 2 of 4)

Some Methods in the Classes JButton, JMenuItem, **and** JLabel

```
public JButton(String text, ImageIcon picture)
public JMenuItem(String text, ImageIcon picture)
public JLabel(
        String text, ImageIcon picture, int horizontalAlignment)
```
Creates a button, menu item, or label with both the text and the icon picture on it. horizontal-Alignment is one of the constants SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT, SwingConstants.LEADING, or SwingConstants.TRAILING. The interface SwingConstants is in the javax.swing package.

```
public void setText(String text)
```
Makes text the only text on the button, menu item, or label.

(continued)

Slide 18-33

Some Methods in the Classes **JButton**, **JMenuItem**, and **JLabel** (Part 3 of 4)

Some Methods in the Classes JButton, JMenuItem, **and** JLabel

```
public void setIcon(ImageIcon picture)
```
Makes picture the only icon on the button, menu item, or label.

```
public void setMargin(Insets margin)
```
JButton and JMenuItem have the method setMargin, but JLabel does not.
The method setMargin sets the size of the margin around the text and icon in the button or menu item.
The following special case will work for most simple situations. The int values give the number of pixels
from the edge of the button or menu item to the text and/or icon.

```
public void setMargin(new Insets(
              int top, int left, int bottom, int right))
```
The class Insets is in the java.awt package. (We will not be discussing any other uses for the class Insets.)

(continued)

Slide 18-34

Some Methods in the Classes **JButton**, **JMenuItem**, and **JLabel** (Part 4 of 4)

Some Methods in the Classes JButton, JMenuItem, **and** JLabel

```
public void setVerticalTextPosition(int textPosition)
```
Sets the vertical position of the text relative to the icon. The textPosition should be one of the con-
stants SwingConstants.TOP, SwingConstants.CENTER (the default position), or SwingCon-
stants.BOTTOM.
The interface SwingConstants is in the javax.swing package.

```
public void setHorizontalTextPosition(int textPosition)
```
Sets the horizontal position of the text relative to the icon. The textPosition should be one of the con-
stants SwingConstants.RIGHT, SwingConstants.LEFT, SwingConstants.CENTER, SwingCon-
stants.LEADING, or SwingConstants.TRAILING.
The interface SwingConstants is in the javax.swing package.

Slide 18-35

The **Insets** Class

- Objects of the class **Insets** are used to specify the size of the margin in a button or menu item
  - The arguments given when an **Insets** class object is created are in pixels
  - The **Insets** class is in the package **java.awt**
  public Insets(int top, int left,
  **   int bottom, int right)**

Slide 18-36

Scroll Bars

- When a text area is created, the number of lines that are visible and the number of characters per line are specified as follows:
  JTextArea memoDisplay = new
  **   JTextArea(15, 30);**
- However, it would often be better not to have to set a firm limit on the number of lines or the number of characters per line
 - This can be done by using *scroll bars* with the text area

Scroll Bars

- When using scroll bars, the text is viewed through a *view port* that shows only part of the text at a time
  - A different part of the text may be viewed by using the scroll bars placed along the side and bottom of the view port
- Scroll bars can be added to text areas using the **JScrollPane** class
  - The **JScrollPane** class is in the **javax.swing** package
  - An object of the class **JScrollPane** is like a view port with scroll bars

18-37

View Port for a Text Area
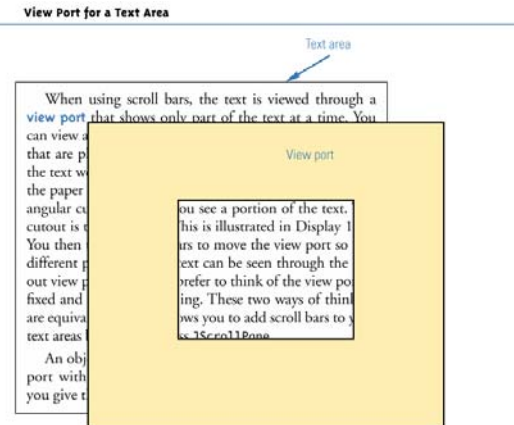
18-38

Scroll Bars

- When a **JScrollPane** is created, the text area to be viewed is given as an argument
  ```
  JScrollPane scrolledText = new
              JScrollPane(memoDisplay);
  ```
- The **JScrollPane** can then be added to a container, such as a **JPanel** or **JFrame**
  ```
  textPanel.add(scrolledText);
  ```

18-39

Scroll Bars

- The scroll bar policies can be set as follows:
  ```
  scrolledText.setHorizontalScrollBarPolicy(
      JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
  scrolledText.setVerticalScrollBarPolicy(
      JscrollPane.VERTICAL_SCROLLBAR_ALWAYS);
  ```
- If invocations of these methods are omitted, then the scroll bars will be visible only when needed
  - If all the text fits in the view port, then no scroll bars will be visible
  - If enough text is added, the scroll bars will appear automatically

18-40

10

Some Methods in the Class `JScrollPane` (Part 1 of 2)

Some Methods in the Class JScrollPane

The JScrollPane class is in the javax.swing package.

```
public JScrollPane(Component objectToBeScrolled)
```

Creates a new JScrollPane for the objectToBeScrolled. Note that the objectToBeScrolled need not be a JTextArea, although that is the only type of argument considered in this book.

```
public void setHorizontalScrollBarPolicy(int policy)
```

Sets the policy for showing the horizontal scroll bar. The policy should be one of

```
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED
```

The phrase AS_NEEDED means the scroll bar is shown only when it is needed. This is explained more fully in the text. The meanings of the other policy constants are obvious from their names.
(As indicated, these constants are defined in the class JScrollPane. You should not need to even be aware of the fact that they have int values. Think of them as policies, not as int values.)

(continued)

18-41

Some Methods in the Class `JScrollPane` (Part 2 of 2)

Some Methods in the Class JScrollPane

```
public void setVerticalScrollBarPolicy(int policy)
```

Sets the policy for showing the vertical scroll bar. The policy should be one of

```
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
JScrollPane.VERTICAL_SCROLLBAR_NEVER
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED
```

The phrase AS_NEEDED means the scroll bar is shown only when it is needed. This is explained more fully in the text. The meanings of the other policy constants are obvious from their names.
(As indicated, these constants are defined in the class JScrollPane. You should not need to even be aware of the fact that they have int values. Think of them as policies, not as int values.)

18-42

A Text Area with Scroll Bars (Part 1 of 8)

A Text Area with Scroll Bars

```
1   import javax.swing.JFrame;
2   import javax.swing.JTextArea;
3   import javax.swing.JPanel;
4   import javax.swing.JLabel;
5   import javax.swing.JButton;
6   import javax.swing.JScrollPane;
7   import java.awt.BorderLayout;
8   import java.awt.FlowLayout;
9   import java.awt.Color;
10  import java.awt.event.ActionListener;
11  import java.awt.event.ActionEvent;
```

(continued)

18-43

A Text Area with Scroll Bars (Part 2 of 8)

A Text Area with Scroll Bars

```
12  public class ScrollBarDemo extends JFrame
13                          implements ActionListener
14  {
15      public static final int WIDTH = 600;
16      public static final int HEIGHT = 400;
17      public static final int LINES = 15;
18      public static final int CHAR_PER_LINE = 30;


19      private JTextArea memoDisplay;
20      private String memo1;
21      private String memo2;
```

(continued)

18-44

A Text Area with Scroll Bars (Part 3 of 8)

A Text Area with Scroll Bars

```
22    public static void main(String[] args)
23    {
24        ScrollBarDemo gui = new ScrollBarDemo();
25        gui.setVisible(true);
26    }


27    public ScrollBarDemo()
28    {
29        super("Scroll Bars Demo");
30        setSize(WIDTH, HEIGHT);
31        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```
                                                        (continued)

A Text Area with Scroll Bars (Part 4 of 8)

A Text Area with Scroll Bars

```
32        JPanel buttonPanel = new JPanel();
33        buttonPanel.setBackground(Color.LIGHT_GRAY);
34        buttonPanel.setLayout(new FlowLayout());
35        JButton memo1Button = new JButton("Save Memo 1");
36        memo1Button.addActionListener(this);
37        buttonPanel.add(memo1Button);

38        JButton memo2Button = new JButton("Save Memo 2");
39        memo2Button.addActionListener(this);
40        buttonPanel.add(memo2Button);

41        JButton clearButton = new JButton("Clear");
42        clearButton.addActionListener(this);
43        buttonPanel.add(clearButton);
```
                                                        (continued)

A Text Area with Scroll Bars (Part 5 of 8)

A Text Area with Scroll Bars

```
44        JButton get1Button = new JButton("Get Memo 1");
45        get1Button.addActionListener(this);
46        buttonPanel.add(get1Button);

47        JButton get2Button = new JButton("Get Memo 2");
48        get2Button.addActionListener(this);
49        buttonPanel.add(get2Button);

50        add(buttonPanel, BorderLayout.SOUTH);

51        JPanel textPanel = new JPanel();
52        textPanel.setBackground(Color.BLUE);
```
                                                        (continued)

A Text Area with Scroll Bars (Part 6 of 8)

A Text Area with Scroll Bars

```
53        memoDisplay = new JTextArea(LINES, CHAR_PER_LINE);
54        memoDisplay.setBackground(Color.WHITE);

55        JScrollPane scrolledText = new JScrollPane(memoDisplay);
56        scrolledText.setHorizontalScrollBarPolicy(
57                JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
58        scrolledText.setVerticalScrollBarPolicy(
59                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

60        textPanel.add(scrolledText);

61        add(textPanel, BorderLayout.CENTER);
62    }
```
                                                        (continued)

A Text Area with Scroll Bars (Part 7 of 8)

A Text Area with Scroll Bars

```
63      public void actionPerformed(ActionEvent e)
64      {
65          String actionCommand = e.getActionCommand();

66          if (actionCommand.equals("Save Memo 1"))
67              memo1 = memoDisplay.getText();
68          else if (actionCommand.equals("Save Memo 2"))
69              memo2 = memoDisplay.getText();
70          else if (actionCommand.equals("Clear"))
71              memoDisplay.setText("");
72          else if (actionCommand.equals("Get Memo 1"))
73              memoDisplay.setText(memo1);
74          else if (actionCommand.equals("Get Memo 2"))
75              memoDisplay.setText(memo2);
76          else
77              memoDisplay.setText("Error in memo interface");
78      }
79  }
```

(continued)

18-49

A Text Area with Scroll Bars (Part 8 of 8)

A Text Area with Scroll Bars

RESULTING GUI

18-50

Components with 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
- A component can be made invisible without making the entire GUI invisible

18-51

Labels with Changing Visibility (Part 1 of 6)

Labels with Changing Visibility

```
1   import javax.swing.JFrame;
2   import javax.swing.ImageIcon;
3   import javax.swing.JPanel;
4   import javax.swing.JLabel;
5   import javax.swing.JButton;
6   import java.awt.BorderLayout;
7   import java.awt.FlowLayout;
8   import java.awt.Color;
9   import java.awt.event.ActionListener;
10  import java.awt.event.ActionEvent;

11  public class VisibilityDemo extends JFrame
12                      implements ActionListener
13  {
14      public static final int WIDTH = 300;
15      public static final int HEIGHT = 200;
```

(continued)

18-52

Labels with Changing Visibility (Part 2 of 6)

Labels with Changing Visibility

```
16    private JLabel wavingLabel;
17    private JLabel standingLabel;
18    public static void main(String[] args)
19    {
20        VisibilityDemo demoGui = new VisibilityDemo();
21        demoGui.setVisible(true);
22    }

23    public VisibilityDemo()
24    {
25        setSize(WIDTH, HEIGHT);
26        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27        setTitle("Visibility Demonstration");

28        setLayout(new BorderLayout());
```
(continued)

18-53

Labels with Changing Visibility (Part 3 of 6)

Labels with Changing Visibility

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

32        ImageIcon dukeStandingIcon =
33                new ImageIcon("duke_standing.gif");
34        standingLabel = new JLabel(dukeStandingIcon);
35        standingLabel.setVisible(true);
36        picturePanel.add(standingLabel);

37        ImageIcon dukeWavingIcon = new ImageIcon("duke_waving.gif");
38        wavingLabel = new JLabel(dukeWavingIcon);
39        wavingLabel.setVisible(false);
40        picturePanel.add(wavingLabel);
```
(continued)

18-54

Labels with Changing Visibility (Part 4 of 6)

Labels with Changing Visibility

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

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

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

48        JButton stopButton = new JButton("Stop");
49        stopButton.addActionListener(this);
50        buttonPanel.add(stopButton);
```
(continued)

18-55

Labels with Changing Visibility (Part 5 of 6)

Labels with Changing Visibility

```
51        add(buttonPanel, BorderLayout.SOUTH);
52    }
53    public void actionPerformed(ActionEvent e)
54    {
55        String actionCommand = e.getActionCommand();

56        if (actionCommand.equals("Wave"))
57        {
58            wavingLabel.setVisible(true);
59            standingLabel.setVisible(false);
60        }
61        else if (actionCommand.equals("Stop"))
62        {
63            standingLabel.setVisible(true);
64            wavingLabel.setVisible(false);
65        }
66        else
67            System.out.println("Unanticipated error.");
68    }
69 }
```
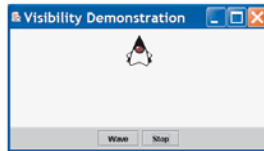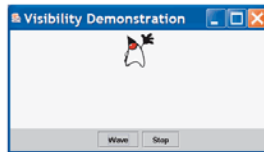(continued)

18-56

Labels with Changing Visibility (Part 6 of 6)

Labels with Changing Visibility

RESULTING GUI (After clicking Stop button)

RESULTING GUI (After clicking Wave button)

18-57

Coordinate System for Graphics Objects

- When drawing objects on the screen, Java uses a coordinate system where the origin point (0,0) is at the upper-left corner of the screen area used for drawing
  - The x-coordinate (horizontal) is positive and increasing to the right
  - The y- coordinate(vertical) is positive and increasing down
  - All coordinates are normally positive
  - Units and sizes are in pixels
  - The area used for drawing is typically a `JFrame` or `JPanel`

18-58

Coordinate System for Graphics Objects

- The point `(x,y)` is located `x` pixels in from the left edge of the screen, and down `y` pixels from the top of the screen
- When placing a rectangle on the screen, the location of its upper-left corner is specified
- When placing a figure other than a rectangle on the screen, Java encloses the figure in an imaginary rectangle, called a *bounding box*, and positions the upper-left corner of this rectangle

18-59

Screen Coordinate System
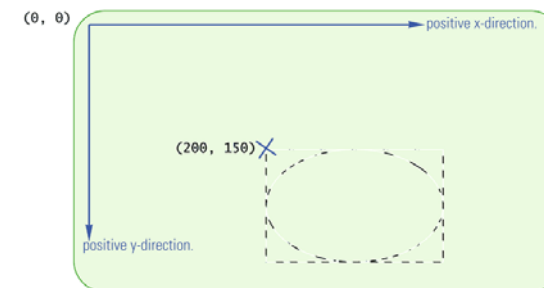
18-60

2014/2/6

The Method **paint** and the Class **Graphics**

- Almost all Swing and Swing-related components and containers have a method called **paint**
- The method **paint** draws the component or container on the screen
  - It is already defined, and is called automatically when the figure is displayed on the screen
  - However, it must be redefined in order to draw geometric figures like circles and boxes
  - When redefined, always include the following:

    super.paint(g);

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.
18-61

The Method **paint** and the Class **Graphics**

- Every container and component that can be drawn on the screen has an associated **Graphics** object
  - The **Graphics** class is an abstract class found in the **java.awt** package
- This object has data specifying what area of the screen the component or container covers
  - The **Graphics** object for a **JFrame** specifies that drawing takes place inside the borders of the **JFrame** object

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.
18-62

The Method **paint** and the Class **Graphics**

- The object **g** of the class **Graphics** can be used as the calling object for a drawing method
  - The drawing will then take place inside the area of the screen specified by **g**
- The method **paint** has a parameter **g** of type **Graphics**
  - When the **paint** method is invoked, **g** is replaced by the **Graphics** object associated with the **JFrame**
  - Therefore, the figures are drawn inside the **JFrame**

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.
18-63

Drawing a Very Simple Face (part 1 of 5)

Drawing a Very Simple Face

```
1   import javax.swing.JFrame;
2   import java.awt.Graphics;
3   import java.awt.Color;

4   public class Face extends JFrame
5   {
6       public static final int WINDOW_WIDTH = 400;
7       public static final int WINDOW_HEIGHT = 400;

8       public static final int FACE_DIAMETER = 200;
9       public static final int X_FACE = 100;
10      public static final int Y_FACE = 100;
```
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.
18-64

16

Drawing a Very Simple Face (part 2 of 5)

Drawing a Very Simple Face

```
11      public static final int EYE_WIDTH = 20;
12      public static final int X_RIGHT_EYE = X_FACE + 55;
13      public static final int Y_RIGHT_EYE = Y_FACE + 60;
14      public static final int X_LEFT_EYE = X_FACE + 130;
15      public static final int Y_LEFT_EYE = Y_FACE + 60;

16      public static final int MOUTH_WIDTH = 100;
17      public static final int X_MOUTH = X_FACE + 50;
18      public static final int Y_MOUTH = Y_FACE + 150;
```

(continued)

Drawing a Very Simple Face (part 3 of 5)

Drawing a Very Simple Face

```
19      public static void main(String[] args)
20      {
21          Face drawing = new Face();
22          drawing.setVisible(true);
23      }

24      public Face()
25      {
26          super("First Graphics Demo");
27          setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
28          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29          getContentPane().setBackground(Color.white);
30      }
```

(continued)

Drawing a Very Simple Face (part 4 of 5)

Drawing a Very Simple Face

```
31      public void paint(Graphics g)
32      {
33          super.paint(g);
34          g.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
35          //Draw Eyes:
36          g.drawLine(X_RIGHT_EYE, Y_RIGHT_EYE,
37                          X_RIGHT_EYE + EYE_WIDTH,Y_RIGHT_EYE);
38          g.drawLine(X_LEFT_EYE, Y_LEFT_EYE,
39                          X_LEFT_EYE + EYE_WIDTH, Y_LEFT_EYE);
40          //Draw Mouth:
41          g.drawLine(X_MOUTH, Y_MOUTH, X_MOUTH + MOUTH_WIDTH, Y_MOUTH);
42      }
43  }
```
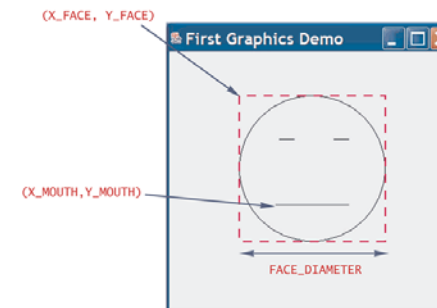
(continued)

Drawing a Very Simple Face (part 5 of 5)

Drawing a Very Simple Face

Some Methods in the Class `Graphics` (part 1 of 4)

Some Methods in the Class Graphics

Graphics is an abstract class in the java.awt package.
Although many of these methods are abstract, we always use them with objects of a concrete descendent class of Graphics, even though we usually do not know the name of that concrete class.

```
public abstract void drawLine(int x1, int y1, int x2, int y2)
```

Draws a line between points (x1, y1) and (x2, y2).

```
public abstract void drawRect(int x, int y,
                               int width, int height)
```

Draws the outline of the specified rectangle. (x, y) is the location of the upper-left corner of the rectangle.

```
public abstract void fillRect(int x, int y,
                               int width, int height)
```

Fills the specified rectangle. (x, y) is the location of the upper-left corner of the rectangle.

(continued)

18-69

Some Methods in the Class `Graphics` (part 2 of 4)

Some Methods in the Class Graphics

```
public void draw3DRect(int x, int y, int width,
                        int height, boolean raised)
```

Draws the outline of the specified rectangle. (x, y) is the location of the upper-left corner. The rectangle is highlighted to look like it has thickness. If raised is true, the highlight makes the rectangle appear to stand out from the background. If raised is false, the highlight makes the rectangle appear to be sunken into the background.

```
public void fill3DRect(int x, int y, int width,
                        int height, boolean raised)
```

Fills the rectangle specified by

```
draw3DRec(x, y, width, height, raised)
```

(continued)

18-70

Some Methods in the Class `Graphics` (part 3 of 4)

Some Methods in the Class Graphics

```
public abstract void drawRoundRect(int x, int y,
          int width, int height, int arcWidth, int arcHeight)
```

Draws the outline of the specified round-cornered rectangle. (x, y) is the location of the upper-left corner of the enclosing regular rectangle. arcWidth and arcHeight specify the shape of the round corners. See the text for details.

```
public abstract void fillRoundRect(int x, int y,
          int width, int height, int arcWidth, int arcHeight)
```

Fills the rounded rectangle specified by

```
drawRoundRec(x, y, width, height, arcWidth, arcHeight)
```

```
public abstract void drawOval(int x, int y,
                               int width, int height)
```

Draws the outline of the oval with the smallest enclosing rectangle that has the specified width and height. The (imagined) rectangle has its upper-left corner located at (x, y).

(continued)

18-71

Some Methods in the Class `Graphics` (part 4 of 4)

Some Methods in the Class Graphics

```
public abstract void fillOval(int x, int y,
                               int width, int height)
```

Fills the oval specified by

```
drawOval(x, y, width, height)
```

```
public abstract void drawArc(int x, int y,
                              int width, int height,
                              int startAngle, int arcSweep)
```

Draws part of an oval that just fits into an invisible rectangle described by the first four arguments. The portion of the oval drawn is given by the last two arguments. See the text for details.

```
public abstract void fillArc(int x, int y,
                              int width, int height,
                              int startAngle, int arcSweep)
```

Fills the partial oval specified by

```
drawArc(x, y, width, height, startAngle, arcSweep)
```

18-72

Drawing Ovals

- An oval is drawn by the method **drawOval**
  - The arguments specify the location, width, and height of the smallest rectangle that can enclose the oval

    g.drawOval(100, 50, 300, 200);
- A circle is a special case of an oval in which the width and height of the rectangle are equal

  g.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);

18-73

Drawing a Happy Face (Part 1 of 5)

Drawing a Happy Face

```
1   import javax.swing.JFrame;
2   import java.awt.Graphics;
3   import java.awt.Color;

4   public class HappyFace extends JFrame
5   {
6       public static final int WINDOW_WIDTH = 400;
7       public static final int WINDOW_HEIGHT = 400;

8       public static final int FACE_DIAMETER = 200;
9       public static final int X_FACE = 100;
10      public static final int Y_FACE = 100;
```
(continued)

18-74

Drawing a Happy Face (Part 2 of 5)

Drawing a Happy Face

```
11      public static final int EYE_WIDTH = 20;
12      public static final int EYE_HEIGHT = 10;
13      public static final int X_RIGHT_EYE = X_FACE + 55;
14      public static final int Y_RIGHT_EYE = Y_FACE + 60;
15      public static final int X_LEFT_EYE = X_FACE + 130;
16      public static final int Y_LEFT_EYE = Y_FACE + 60;

17      public static final int MOUTH_WIDTH = 100;
18      public static final int MOUTH_HEIGHT = 50;
19      public static final int X_MOUTH = X_FACE + 50;
20      public static final int Y_MOUTH = Y_FACE + 100;
21      public static final int MOUTH_START_ANGLE = 180;
22      public static final int MOUTH_ARC_SWEEP = 180;
```
(continued)

18-75

Drawing a Happy Face (Part 3 of 5)

Drawing a Happy Face

```
23      public static void main(String[] args)
24      {
25          HappyFace drawing = new HappyFace();
26          drawing.setVisible(true);
27      }

28      public HappyFace()
29      {
30          super("Graphics Demonstration 2");
31          setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
32          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33          getContentPane().setBackground(Color.white);
34      }
```
(continued)

18-76

Drawing a Happy Face (Part 4 of 5)

Drawing a Happy Face

```
35    public void paint(Graphics g)
36    {
37        super.paint(g);
38        g.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
39        //Draw Eyes:
40        g.fillOval(X_RIGHT_EYE, Y_RIGHT_EYE, EYE_WIDTH, EYE_HEIGHT);
41        g.fillOval(X_LEFT_EYE, Y_LEFT_EYE, EYE_WIDTH, EYE_HEIGHT);
42        //Draw Mouth:
43        g.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH, MOUTH_HEIGHT,
44                  MOUTH_START_ANGLE, MOUTH_ARC_SWEEP);
45    }
46 }
```

(continued)

18-77

Drawing a Happy Face (Part 5 of 5)

Drawing a Happy Face

RESULTING GUI
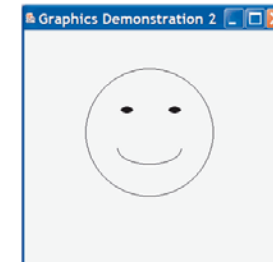
18-78

Drawing Arcs

- Arcs are described by giving an oval, and then specifying a portion of it to be used for the arc
  - The following statement draws the smile on the happy face:
    ```
    g.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH,
        MOUTH_HEIGHT, MOUTH_START_ANGLE,
        MOUTH_ARC_SWEEP);
    ```
  - The arguments `MOUTH_WIDTH` and `MOUTH_HEIGHT` determine the size of the bounding box, while the arguments `X_MOUTH` and `Y_MOUTH` determine its location
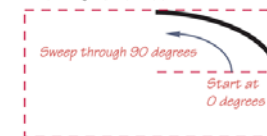 - The last two arguments specify the portion made visible

18-79

Specifying an Arc (Part 1 of 2)

Specifying an Arc

```
g.drawArc(x, y, width, height, 0, 90);
```

Sweep through 90 degrees

Start at 0 degrees

```
g.drawArc(x, y, width, height, 0, -90);
```

Start at 0 degrees

Negative direction
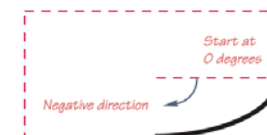
(continued)

18-80

Specifying an Arc (Part 2 of 2)
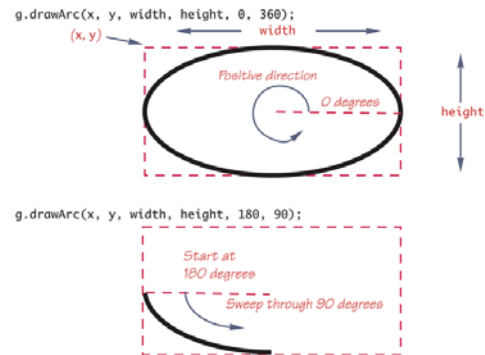
18-81

## Rounded Rectangles

- A *rounded rectangle* is a rectangle whose corners have been replaced by arcs so that the corners are rounded

  `g.drawRoundRect(x, y, width, height,`
  `                             arcWidth, arcHeight)`

  - The arguments **x**, **y**, **width**, and **height** determine a regular rectangle in the usual way
  - The last two arguments **arcWidth**, and **arcHeight**, specify the arcs that will be used for the corners
  - Each corner is replaced with an quarter of an oval that is **arcWidth** pixels wide and **arcHeight** pixels high
  - When **arcWidth** and **arcHeight** are equal, the corners will be arcs of circles

18-82

A Rounded Rectangle
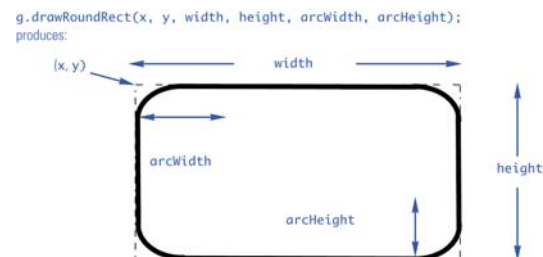
18-83

paintComponent for Panels

- A **JPanel** is a **JComponent**, but a **JFrame** is a **Component**, not a **JComponent**
  - Therefore, they use different methods to paint the screen
- Figures can be drawn on a **JPanel**, and the **JPanel** can be placed in a **JFrame**
  - When defining a **JPanel** class in this way, the **paintComponent** method is used instead of the **paint** method
  - Otherwise the details are the same as those for a **JFrame**

18-84

21

paintComponent Demonstration (Part 1 of 4)

paintComponent **Demonstration**

```
1   import javax.swing.JFrame;
2   import javax.swing.JPanel;
3   import java.awt.GridLayout;
4   import java.awt.Graphics;
5   import java.awt.Color;

6   public class PaintComponentDemo extends JFrame
7   {
8       public static final int FRAME_WIDTH = 400;
9       public static final int FRAME_HEIGHT = 400;
```
(continued)

paintComponent Demonstration (Part 2 of 4)

paintComponent **Demonstration**

```
10      private class FancyPanel extends JPanel
11      {
12          public void paintComponent(Graphics g)
13          {
14              super.paintComponent(g);
15              setBackground(Color.YELLOW);
16              g.drawOval(FRAME_WIDTH/4, FRAME_HEIGHT/8,
17                         FRAME_WIDTH/2, FRAME_HEIGHT/6);
18          }
19      }

20      public static void main(String[] args)
21      {
```
(continued)

paintComponent Demonstration (Part 3 of 4)

paintComponent **Demonstration**

```
22          PaintComponentDemo w = new PaintComponentDemo();
23          w.setVisible(true);
24      }
25      public PaintComponentDemo()
26      {
27          setSize(FRAME_WIDTH, FRAME_HEIGHT);
28          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29          setTitle("The Oval Is in a Panel");
30          setLayout(new GridLayout(2, 1));
31          FancyPanel p = new FancyPanel();
32          add(p);
33          JPanel whitePanel = new JPanel();
34          whitePanel.setBackground(Color.WHITE);
35          add(whitePanel);
36      }
37  }
```
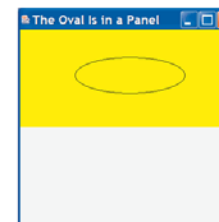(continued)

paintComponent **Demonstration**

RESULTING GUI

Action Drawings and `repaint`

- The `repaint` method should be invoked when the graphics content of a window is changed
  - The `repaint` method takes care of some overhead, and then invokes the method `paint`, which redraws the screen
  - Although the `repaint` method must be explicitly invoked, it is already defined
  - The `paint` method, in contrast, must often be defined, but is not explicitly invoked

18-89

An Action Drawing (Part 1 of 7)

An Action Drawing

```
1   import javax.swing.JFrame;
2   import javax.swing.JButton;
3   import java.awt.event.ActionListener;
4   import java.awt.event.ActionEvent;
5   import java.awt.BorderLayout;
6   import java.awt.Graphics;
7   import java.awt.Color;

8   public class ActionFace extends JFrame
9   {
10      public static final int WINDOW_WIDTH = 400;
11      public static final int WINDOW_HEIGHT = 400;
```
(continued)

18-90

An Action Drawing (Part 2 of 7)

An Action Drawing

```
12      public static final int FACE_DIAMETER = 200;
13      public static final int X_FACE = 100;
14      public static final int Y_FACE = 100;

15      public static final int EYE_WIDTH = 20;
16      public static final int EYE_HEIGHT = 10;
17      public static final int X_RIGHT_EYE = X_FACE + 55;
18      public static final int Y_RIGHT_EYE = Y_FACE + 60;
19      public static final int X_LEFT_EYE = X_FACE + 130;
20      public static final int Y_LEFT_EYE = Y_FACE + 60;
```
(continued)

18-91

An Action Drawing (Part 3 of 7)

An Action Drawing

```
21      public static final int MOUTH_WIDTH = 100;
22      public static final int MOUTH_HEIGHT = 50;
23      public static final int X_MOUTH = X_FACE + 50;
24      public static final int Y_MOUTH = Y_FACE + 100;
25      public static final int MOUTH_START_ANGLE = 180;
26      public static final int MOUTH_ARC_SWEEP = 180;

27      private boolean wink;

28      private class WinkAction implements ActionListener
29      {
30          public void actionPerformed(ActionEvent e)
31          {
32              wink = true;
33              repaint();
34          }
35      } // End of WinkAction inner class
```
(continued)

18-92

An Action Drawing (Part 4 of 7)

An Action Drawing

```
36      public static void main(String[] args)
37      {
38          ActionFace drawing = new ActionFace();
39          drawing.setVisible(true);
40      }

41      public ActionFace()
42      {
43          setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
44          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
45          setTitle("Hello There!");
46          setLayout(new BorderLayout());
47          getContentPane().setBackground(Color.white);

48          JButton winkButton = new JButton("Click for a Wink.");
49          winkButton.addActionListener(new WinkAction());
50          add(winkButton, BorderLayout.SOUTH);
51          wink = false;
52      }
```

(continued)

18-93

An Action Drawing (Part 5 of 7)

An Action Drawing

```
53      public void paint(Graphics g)
54      {
55          super.paint(g);
56          g.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
57          //Draw Right Eye:
58          g.fillOval(X_RIGHT_EYE, Y_RIGHT_EYE, EYE_WIDTH, EYE_HEIGHT);
59          //Draw Left Eye:
60          if (wink)
61              g.drawLine(X_LEFT_EYE, Y_LEFT_EYE,
62                                      X_LEFT_EYE + EYE_WIDTH, Y_LEFT_EYE);
63          else
64              g.fillOval(X_LEFT_EYE, Y_LEFT_EYE, EYE_WIDTH, EYE_HEIGHT);
65          //Draw Mouth:
66          g.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH, MOUTH_HEIGHT,
67                      MOUTH_START_ANGLE, MOUTH_ARC_SWEEP);
68      }

69  }
```

(continued)

18-94

An Action Drawing (Part 6 of 7)

An Action Drawing

RESULTING GUI (When started)

(continued)

18-95

An Action Drawing (Part 7 of 7)

An Action Drawing

RESULTING GUI (After clicking the button)

18-96

24
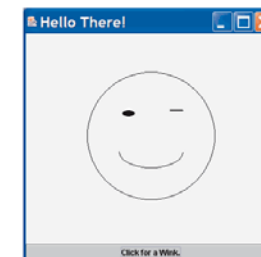
Some More Details on Updating a GUI

- With Swing, most changes to a GUI are updated automatically to become visible on the screen
  - This is done by the *repaint manager* object
- Although the repaint manager works automatically, there are a few updates that it does not perform
  - For example, the ones taken care of by **validate** or **repaint**
- One other updating method is **pack**
  - **pack** resizes the window to something known as the preferred size

The **validate** Method

- An invocation of **validate** causes a container to lay out its components again
  - It is a kind of "update" method that makes changes in the components shown on the screen
  - Every container class has the **validate** method, which has no arguments
- Many simple changes made to a Swing GUI happen automatically, while others require an invocation of **validate** or some other "update" method
  - When in doubt, it will do no harm to invoke **validate**

Specifying a Drawing Color

- Using the method **drawLine** inside the **paint** method is similar to drawing with a pen that can change colors
  - The method **setColor** will change the color of the pen
  - The color specified can be changed later on with another invocation of **setColor** so that a single drawing can have multiple colors

  g.setColor(Color.BLUE)

Adding Color

Adding Color

```
1     public void paint(Graphics g)
2     {
3         super.paint(g);
4         //Default is equivalent to: g.setColor(Color.black);
5         g.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
6         //Draw Eyes:
7         g.setColor(Color.BLUE);
8         g.fillOval(X_RIGHT_EYE, Y_RIGHT_EYE, EYE_WIDTH, EYE_HEIGHT);
9         g.fillOval(X_LEFT_EYE, Y_LEFT_EYE, EYE_WIDTH, EYE_HEIGHT);
10        //Draw Mouth:
11        g.setColor(Color.RED);
12        g.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH, MOUTH_HEIGHT,
13                  MOUTH_START_ANGLE, MOUTH_ARC_SWEEP);
14    }
```

If you replace the **paint** method in Display 18.13 with this version then the happy face will have blue eyes and red lips.

Defining Colors

- Standard colors in the class **Color** are already defined in Chapter 17
- The **Color** class can also be used to define additional colors
  - It uses the *RGB color system* in which different amounts of red, green, and blue light are used to produce any color

18-101

The Color Constants

The Color Constants

```
Color.BLACK          Color.MAGENTA
Color.BLUE           Color.ORANGE
Color.CYAN           Color.PINK
Color.DARK_GRAY      Color.RED
Color.GRAY           Color.WHITE
Color.GREEN          Color.YELLOW
Color.LIGHT_GRAY
```

18-102

Defining Colors

- Integers or floats may be used when specifying the amount of red, green, and/or blue in a color
  - Integers must be in the range 0-255 inclusive
    ```
    Color brown = new Color(200, 150, 0);
    ```
  - **float** values must be in the range 0.0-1.0 inclusive
    ```
    Color brown = new Color(
       (float)(200.0/255),(float)(150.0/255),
                                (float)0.0);
    ```

18-103

Pitfall: Using **double**s to Define a Color

- Constructors for the class **Color** only accept arguments of type **int** or **float**
  - Without a cast, numbers like 200.0/255, 0.5, and 0.0 are considered to be of type **double**, not of type **float**
- Don't forget to use a type cast when intending to use **float** numbers
  - Note that these numbers should be replaced by defined constants in any final code produced
    ```
    public static final float RED_VALUE =
                                (float)0.5;
    ```

18-104

Some Methods in the Class `Color` (Part 1 of 2)

Some Methods in the Class Color

The class Color is in the java.awt package.

    public Color(int r, int g, int b)

Constructor that creates a new Color with the specified RGB values. The parameters r, g, and b must each
be in the range 0 to 255 (inclusive).

    public Color(float r, float g, float b)

Constructor that creates a new Color with the specified RGB values. The parameters r, g, and b must each
be in the range 0.0 to 1.0 (inclusive).

    public int getRed()

Returns the red component of the calling object. The returned value is in the range 0 to 255 (inclusive).

(continued)

18-105

Some Methods in the Class `Color` (Part 2 of 2)

Some Methods in the Class Color

    public int getGreen()

Returns the green component of the calling object. The returned value is in the range 0 to 255 (inclusive).

    public int getBlue()

Returns the blue component of the calling object. The returned value is in the range 0 to 255 (inclusive).

    public Color brighter()

Returns a brighter version of the calling object color.

    public Color darker()

Returns a darker version of the calling object color.

    public boolean equals(Object c)

Returns true if c is equal to the calling object color; otherwise, returns false.

18-106

The `JColorChooser` Dialog Window

- The class **JColorChooser** can be used to allow a user to choose a color
- The **showDialog** method of **JColorChooser** produces a color-choosing window
  - The user can choose a color by selecting RGB values or from a set of color samples

```
sample Color =
          JColorChooser.showDialog(this,
          "JColorChooser", sampleColor);
```

18-107

`JColorChooser` Dialog (Part 1 of 5)

JColorChooser **Dialog**

```
1   import javax.swing.JFrame;
2   import javax.swing.JPanel;
3   import javax.swing.JButton;
4   import javax.swing.JColorChooser;
5   import java.awt.event.ActionListener;
6   import java.awt.event.ActionEvent;
7   import java.awt.BorderLayout;
8   import java.awt.FlowLayout;
9   import java.awt.Color;
```

(continued)

18-108

JColorChooser Dialog (Part 2 of 5)

JColorChooser **Dialog**

```
10   public class JColorChooserDemo extends JFrame
11                              implements ActionListener
12   {
13       public static final int WIDTH = 400;
14       public static final int HEIGHT = 200;

15       private Color sampleColor = Color.LIGHT_GRAY;

16       public static void main(String[] args)
17       {
18           JColorChooserDemo gui = new JColorChooserDemo();
19           gui.setVisible(true);
20       }
```

(continued)

JColorChooser Dialog (Part 3 of 5)

JColorChooser **Dialog**

```
21       public JColorChooserDemo()
22       {
23           setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24           getContentPane().setBackground(sampleColor);
25           setLayout(new BorderLayout());
26           setTitle("JColorChooser Demo");
27           setSize(WIDTH, HEIGHT);
28           JPanel buttonPanel = new JPanel();
29           buttonPanel.setBackground(Color.WHITE);
30           buttonPanel.setLayout(new FlowLayout());
31           JButton chooseButton = new JButton("Choose a Color");
32           chooseButton.addActionListener(this);
33           buttonPanel.add(chooseButton);
34           add(buttonPanel, BorderLayout.SOUTH);
35       }
```

(continued)

JColorChooser Dialog (Part 4 of 5)

JColorChooser **Dialog**

```
36       public void actionPerformed(ActionEvent e)
37       {
38           if (e.getActionCommand().equals("Choose a Color"))
39           {
40               sampleColor =
41                   JColorChooser.showDialog(this, "JColorChooser", sampleColor);
42               if (sampleColor != null)//If a color was chosen
43                   getContentPane().setBackground(sampleColor);
44           }
45           else
46               System.out.println("Unanticipated Error");
47       }
48   }
```
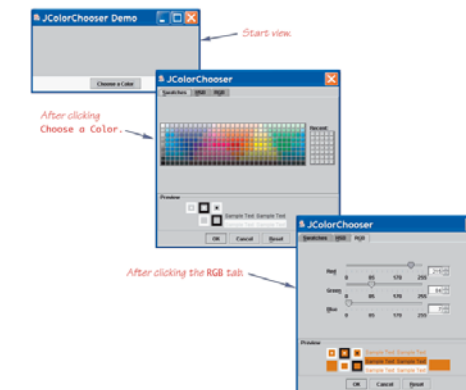
(continued)

JColorChooser Dialog (Part 5 of 5)

JColorChooser **Dialog**

RESULTING GUI (Three views of one GUI)

The `drawString` Method

- The method **drawString** is similar to the drawing methods in the **Graphics** class
  - However, it displays text instead of a drawing
  - If no font is specified, a default font is used

`g.drawString(theText, X_START, Y_Start);`

18-113

Using `drawString` (Part 1 of 7)

Using drawString

```
1   import javax.swing.JFrame;
2   import javax.swing.JPanel;
3   import javax.swing.JButton;
4   import java.awt.event.ActionListener;
5   import java.awt.event.ActionEvent;
6   import java.awt.BorderLayout;
7   import java.awt.Graphics;
8   import java.awt.Color;
9   import java.awt.Font;
```

(continued)

18-114

Using `drawString` (Part 2 of 7)

Using drawString

```
10  public class DrawStringDemo extends JFrame
11                         implements ActionListener
12  {
13      public static final int WIDTH = 350;
14      public static final int HEIGHT = 200;
15      public static final int X_START = 20;
16      public static final int Y_START = 100;
17      public static final int POINT_SIZE = 24;

18      private String theText = "Push the button.";
19      private Color penColor = Color.BLACK;
20      private Font fontObject =
21                  new Font("SansSerif", Font.PLAIN, POINT_SIZE);

22      public static void main(String[] args)
23      {
24          DrawStringDemo gui = new DrawStringDemo();
25          gui.setVisible(true);
26      }
```

(continued)

18-115

Using `drawString` (Part 3 of 7)

Using drawString

```
27      public DrawStringDemo()
28      {
29          setSize(WIDTH, HEIGHT);
30          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31          setTitle("drawString Demonstration");

32          getContentPane().setBackground(Color.WHITE);
33          setLayout(new BorderLayout());

34          JPanel buttonPanel = new JPanel();
35          buttonPanel.setBackground(Color.GRAY);
36          buttonPanel.setLayout(new BorderLayout());
```

(continued)

18-116

2014/2/6

Using **drawString** (Part 4 of 7)

Using *drawString*

```
37          JButton theButton = new JButton("The Button");
38          theButton.addActionListener(this);

39          buttonPanel.add(theButton, BorderLayout.CENTER);

40          add(buttonPanel, BorderLayout.SOUTH);
41      }

42      public void paint(Graphics g)
43      {
44          super.paint(g);
45          g.setFont(fontObject);
46          g.setColor(penColor);
47          g.drawString(theText, X_START, Y_START);
48      }
```

(continued)

18-117

Using **drawString** (Part 5 of 7)

Using *drawString*

```
49      public void actionPerformed(ActionEvent e)
50      {
51          penColor = Color.RED;
52          fontObject =
53              new Font("Serif", Font.BOLD|Font.ITALIC, POINT_SIZE);
54          theText = "Thank you. I needed that.";

55          repaint();
56      }
57  }
```

(continued)

18-118

Using **drawString** (Part 6 of 7)

Using *drawString*

RESULTING GUI (Start view)

(continued)

18-119

Using **drawString** (Part 7 of 7)

Using *drawString*

RESULTING GUI (After clicking the button)

18-120

30

Fonts

- A font is an object of the **Font** class
  - The **Font** class is found in the **java.awt** package
- The constructor for the **Font** class creates a font in a given style and size
  ```
  Font fontObject = new Font("SansSerif",
                      Font.PLAIN, POINT_SIZE);
  ```
- A program can set the font for the **drawString** method within the **paint** method
  ```
  g.setFont(fontObject);
  ```

18-121

Font Types

- Any font currently available on a system can be used in Java
  - However, Java guarantees that at least three fonts will be available: **"Monospaced"**, **"SansSerif"**, and **"Serif"**
- *Serifs* are small lines that finish off the ends of the lines in letters
  - This **S** has serifs, but this S does not
  - A **"Serif"** font will always have serifs
  - Sans means without, so the **"SansSerif"** font will not have serifs
  - **"Monospaced"** means that all the characters have equal width

18-122

Font Styles

- Fonts can be given style modifiers, such as bold or italic
  - Multiple styles can be specified by connecting them with the | symbol (called the bitwise OR symbol)
    ```
    new Font("Serif",
        Font.BOLD|Font.ITALIC, POINT_SIZE);
    ```
- The size of a font is called its *point size*
  - Character sizes are specified in units known as *points*
  - One point is 1/72 of an inch

18-123

Result of Running **FontDisplay.java** (Found on the Accompanying CD)

18-124

Some Methods and Constants for the Class `Font` (Part 1 of 2)

Some Methods and Constants for the Class Font

The class Font is in the java.awt package.

CONSTRUCTOR FOR THE CLASS Font

public Font(String fontName, int styleModifications, int size)

Constructor that creates a version of the font named by fontName with the specified styleModifications and size.

CONSTANTS IN THE CLASS Font

Font.BOLD

Specifies bold style.

Font.ITALIC

Specifies italic style.

Font.PLAIN

Specifies plain style—that is, not bold and not italic.

18-125

Some Methods and Constants for the Class `Font` (Part 2 of 2)

NAMES OF FONTS (These three are guaranteed by Java. Your system will probably have others as well as these.)

"Monospaced"

See Display 18.22 for a sample.

"SansSerif"

See Display 18.22 for a sample.

"Serif"

See Display 18.22 for a sample.

METHOD THAT USES Font

public abstract void setFont(Font fontObject)

This method is in the class Graphics. Sets the current font of the calling Graphics object to fontObject.

18-126