Chapter 6:
Graphical User Interfaces

John Lewis, Peter DePasquale, and Joseph Chase

Addison-Wesley is an imprint of PEARSON

들어가기전에

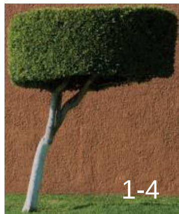
- GUI에서 중요한 구성요소 3가지는 무엇인가? 이들의 관계를 자동문(그림참고)을 예로 들어 설명하시오.
- 창틀, 유리창, 자동문 버튼(유리창에 부착)의 관계를 바탕으로 7쪽과 8쪽의 소스코드를 설명하시오.
- Mouse Event 종류를 설명하시오.
- Key Event 종류를 설명하시오.
- Tool Tips과 Mnemonics을 설명하시오.

Graphical User Interfaces

- Many programs provide graphical user interfaces (GUI) through which a user interacts
- Chapter 6 focuses on
  - discussing the core elements
    - components, events, and listeners
  - discussing the concept of a layout manager
  - examining mouse and keyboard events
  - describing tool tips and mnemonics
  - discussing GUI design issues

Outline

- GUI Elements
- More Components
- Layout Managers
- Mouse and Key Events
- Dialog Boxes
- Important Details
- GUI Design



6.1 – GUI Elements

- Three kinds of objects are needed to create a GUI in Java
  - *Component* – an object that defines a screen element used to display information or allow the user to interact with the program
  - *Event* – an object that represents some occurrence in which we may be interested
    - mouse button press, keyboard key press
  - *Listener* – an object that "waits" for an event to occur and responds in way when it does
- It's common to use existing components and events from the Java class library.
  - We will, however, write our own listener classes to perform whatever actions we desire when events occur

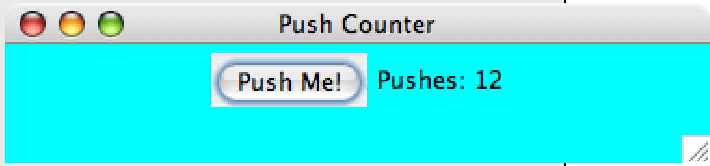
6.1 – GUI Elements

- GUI-oriented program is called *event-driven*
  - Command-line program is called *procedure-driven*
- To create a Java program that uses a GUI, we must:
  - instantiate and set up the necessary components,
  - implement listener classes that define what happens when particular events occur, and
  - establish the relationship between the listeners and the components
- Java components and other GUI-related classes are defined primarily in two packages
  - java.awt
  - javax.swing

6.1 – GUI Elements

- Let’s look at a simple example that contains all of the basic GUI elements
  - the example presents the user with a single push button
  - each time the button is pushed, a counter is updated and displayed

```
//  Demonstrates a graphical user interface and an event listener.
import javax.swing.JFrame;

public class PushCounter
{
   //  Creates and displays the main program frame.
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Push Counter");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      PushCounterPanel panel = new PushCounterPanel();
      frame.getContentPane().add(panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```

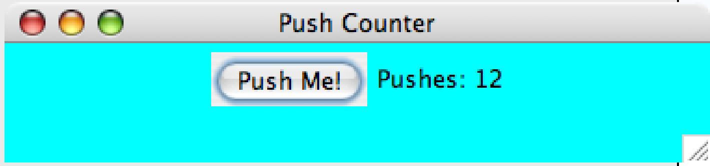


Container managed by OS system

Container managed by Java program

6.1 – PushCounterPanel.java

```
//  Demonstrates a graphical user interface and an event listener.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;

   public PushCounterPanel ()                //  Constructor: Sets up the GUI.
   {
      count = 0;

      push = new JButton ("Push Me!");
      push.addActionListener (new ButtonListener());

      label = new JLabel ("Pushes: " + count);

      add (push);
      add (label);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 40));
   }

   //  Represents a listener for button push (action) events.
   private class ButtonListener implements ActionListener
   {
      //  Updates the counter and label when the button is pushed.
      public void actionPerformed (ActionEvent event)
      {
         count++;
         label.setText("Pushes: " + count);
      }
   }
}
```



Inner Class

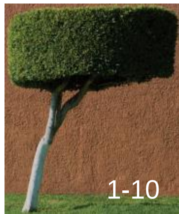
Can access the instance variables `count` and `label`

6.1 – Buttons and Action Events

- `ButtonListener` class was created as an *inner class*
  - Inner classes have access to the members of the class that contains it
- `ButtonListener` implements the `ActionListener` interface
  - The implementing class must implement list of methods
- `ActionListener` interface
  - only method listed is the `actionPerformed` method
  - the button component generates the action event resulting in a call to the `actionPerformed` method, passing an `ActionEvent` object

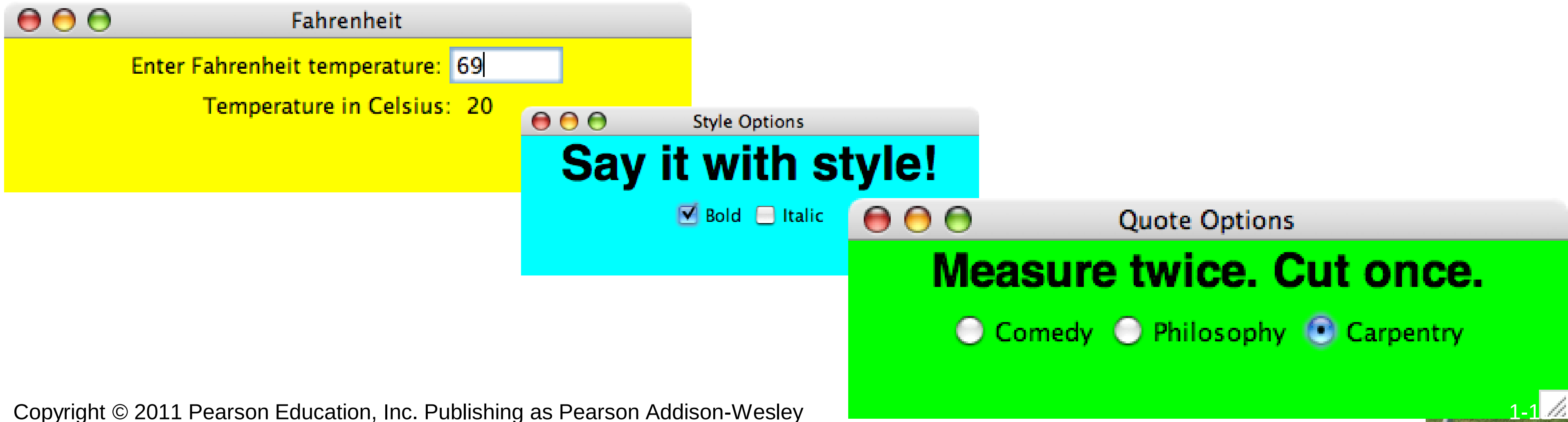
Outline

- GUI Elements
- ⇨ More Components
- Layout Managers
- Mouse and Key Events
- Dialog Boxes
- Important Details
- GUI Design



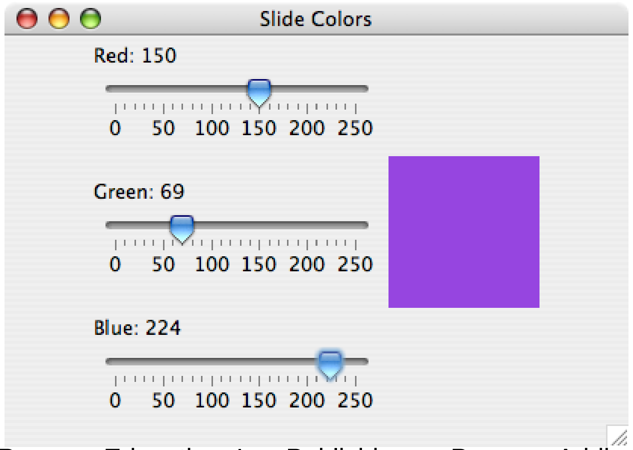
6.2 – More Components

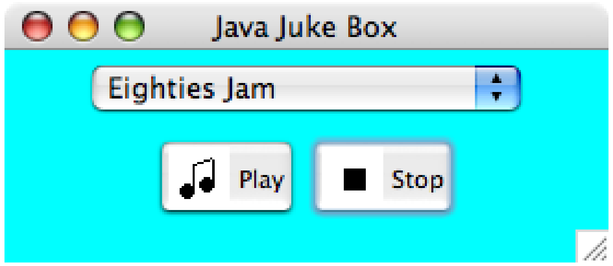
- In addition to push buttons, there are variety of other interactive components
  - *Text fields* allows the user to enter typed input from the keyboard
  - *Check boxes* – a button that can be toggled on or off using the mouse (indicates a boolean value is set or unset)
  - *Radio buttons* used with other radio buttons to provide a set of mutually exclusive options



6.2 – More Components

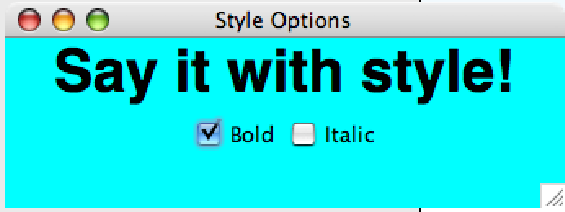
- In addition to push buttons, there are variety of other interactive components
  - *Sliders* allows the user to specify a numeric value within a bounded range
  - *Combo boxes* allows the user to select one of several options from a "drop down" menu
  - *Timers* helps us manage an activity over time, has no visual representation





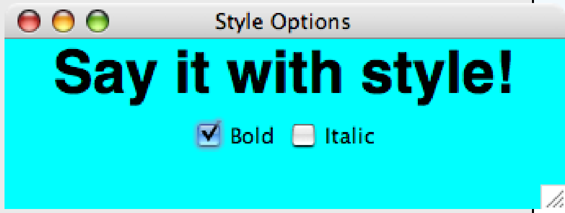
6.2 – StyleOptions.java

```
//  Demonstrates the use of check boxes.

import javax.swing.JFrame;

public class StyleOptions
{
   //  Creates and displays the style options frame.
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Style Options");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new StyleOptionsPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



6.2 – StyleOptionsPanel.java

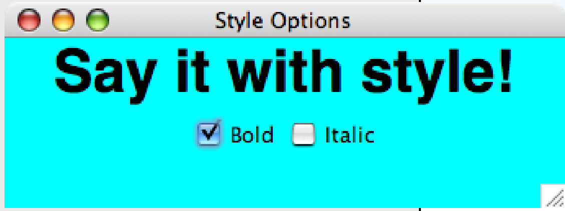
```
//  Demonstrates the use of check boxes.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel
{
   private JLabel saying;
   private JCheckBox bold, italic;

   // Sets up a panel with a label and some check boxes to control the style of the label's font.
   public StyleOptionsPanel()
   {
      saying = new JLabel ("Say it with style!");
      saying.setFont (new Font ("Helvetica", Font.PLAIN, 36));

      bold = new JCheckBox ("Bold");
      bold.setBackground (Color.cyan);
      italic = new JCheckBox ("Italic");
      italic.setBackground (Color.cyan);

      StyleListener listener = new StyleListener();
      bold.addItemListener (listener);
      italic.addItemListener (listener);

      add (saying);
      add (bold);
      add (italic);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 100));
   }
```



(more…)

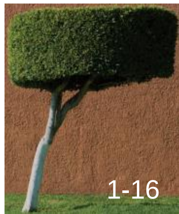
6.2 – StyleOptionsPanel.java

```
   // Represents the listener for both check boxes.
   private class StyleListener implements ItemListener
   {
      // Updates the style of the label font style.
      public void itemStateChanged (ItemEvent event)
      {
         int style = Font.PLAIN;

         if (bold.isSelected())
            style = Font.BOLD;

         if (italic.isSelected())
            style += Font.ITALIC;

         saying.setFont (new Font ("Helvetica", style, 36));
      }
   }
}
```



Outline

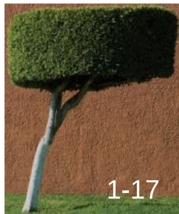
- GUI Elements
- More Components
- ⇨ Layout Managers
- Mouse and Key Events
- Dialog Boxes
- Important Details
- GUI Design



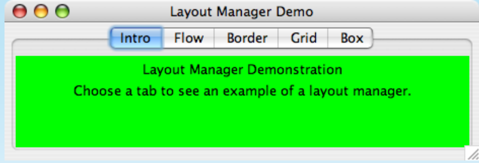
6.3 – Layout Managers

- Every container is managed by an object known as a *layout manager* that determines how the components in the container are arranged visually
  - The layout manager is consulted when needed, such as when the container is resized or when a component is added

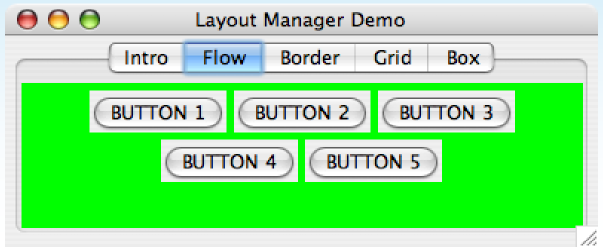
Layout Manager	**Description**
Flow Layout	Organizes components from left to right, starting new rows as necessary
Border Layout	Organizes components into five areas (North, South, East, West, and Center)
Grid Layout	Organizes components into a grid of rows and columns
Box Layout	Organizes components into a single row or column

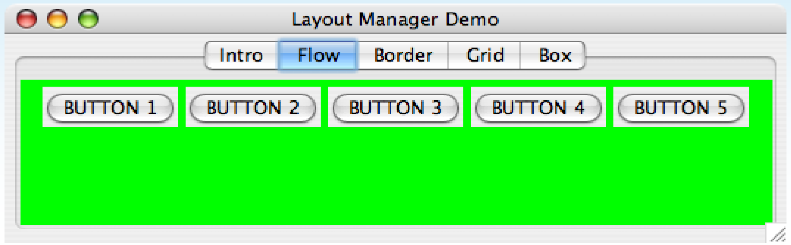


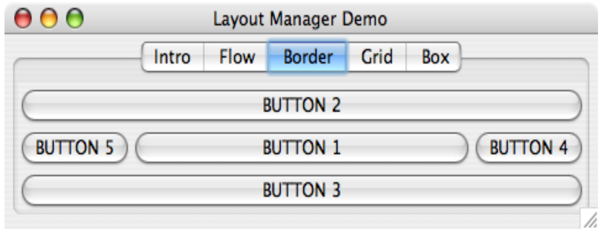
6.3 – Layout Manager Demo

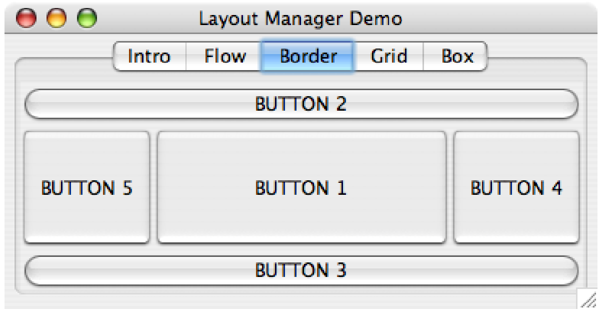


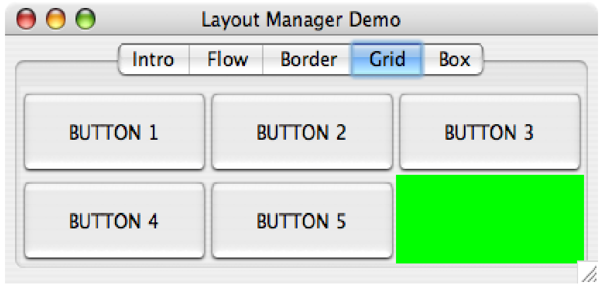
Layout Manager Demonstration
Choose a tab to see an example of a layout manager.

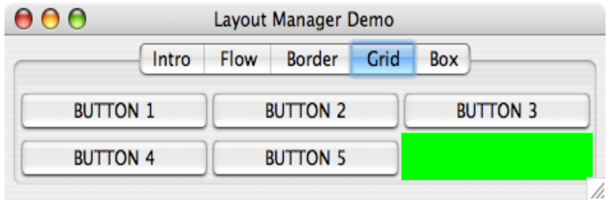


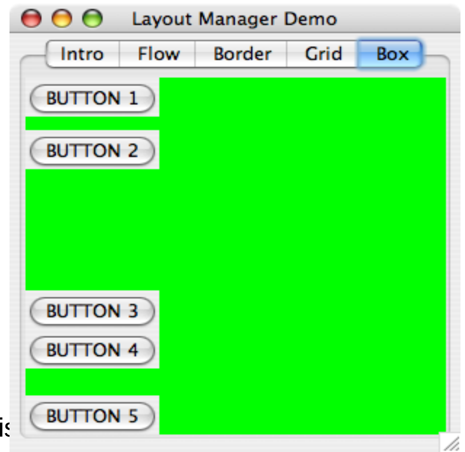


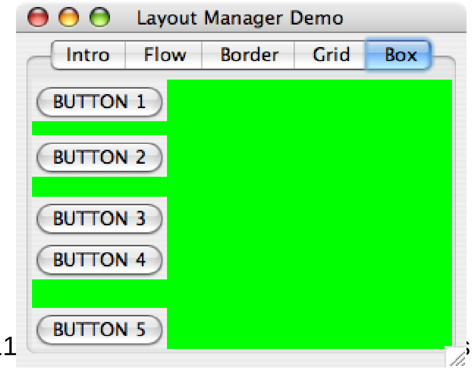


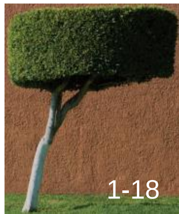






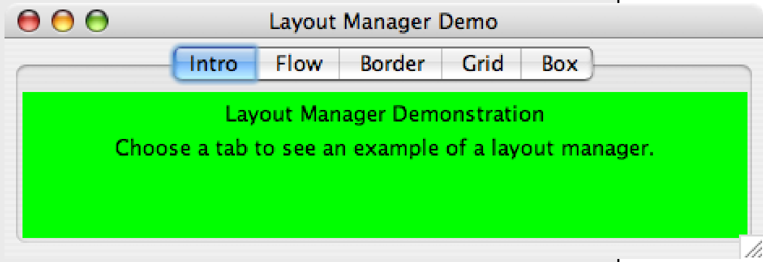






6.3 – LayoutDemo.java

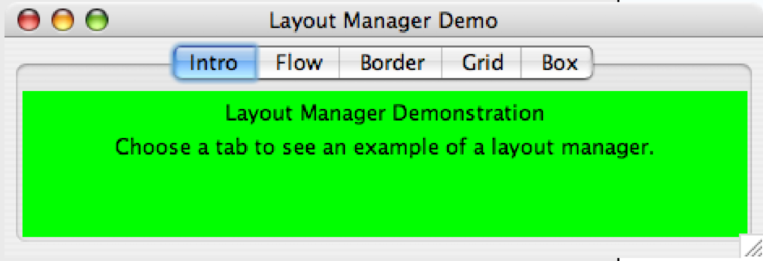
```java
//  Demonstrates the use of flow, border, grid, and box layouts.

import javax.swing.*;

public class LayoutDemo
{
   //  Sets up a frame containing a tabbed pane.
   // The panel on each tab demonstrates a different layout manager.
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Layout Manager Demo");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      JTabbedPane tp = new JTabbedPane();
      tp.addTab ("Intro", new IntroPanel());
      tp.addTab ("Flow", new FlowPanel());
      tp.addTab ("Border", new BorderPanel());
      tp.addTab ("Grid", new GridPanel());
      tp.addTab ("Box", new BoxPanel());

      frame.getContentPane().add(tp);

      frame.pack();
      frame.setVisible(true);
   }
}
```



Layout Manager Demonstration
Choose a tab to see an example of a layout manager.

6.3 – IntroPanel.java

```
//  Represents the introduction panel for the LayoutDemo program.

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

public class IntroPanel extends JPanel
{
   //  Sets up this panel with two labels.
   public IntroPanel()
   {
      setBackground (Color.green);

      JLabel l1 = new JLabel ("Layout Manager Demonstration");
      JLabel l2 = new JLabel ("Choose a tab to see an example of " +
                              "a layout manager.");
      add (l1);
      add (l2);
   }
}
```

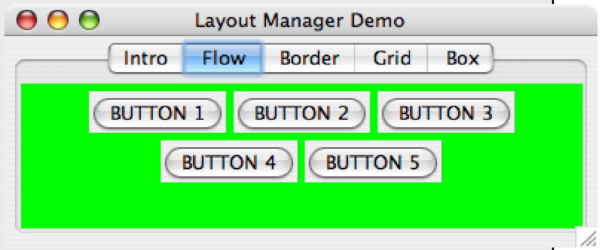


Layout Manager Demonstration

Choose a tab to see an example of a layout manager.

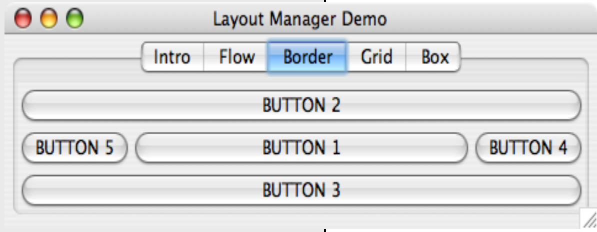
6.3 – FlowPanel.java

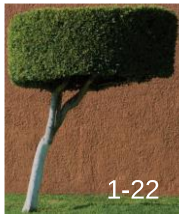
```
// Represents the panel in the LayoutDemo program that demonstrates the flow layout manager.

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

public class FlowPanel extends JPanel
{
   // Sets up this panel with some buttons to show how flow layout affects their position.
   public FlowPanel ()
   {
      setLayout (new FlowLayout());

      setBackground (Color.green);

      JButton b1 = new JButton ("BUTTON 1");
      JButton b2 = new JButton ("BUTTON 2");
      JButton b3 = new JButton ("BUTTON 3");
      JButton b4 = new JButton ("BUTTON 4");
      JButton b5 = new JButton ("BUTTON 5");

      add (b1);
      add (b2);
      add (b3);
      add (b4);
      add (b5);
   }
}
```



6.3 – BorderPanel.java

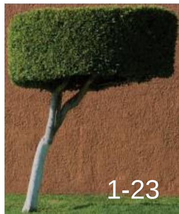
```
// Represents the panel in the LayoutDemo program
// that demonstrates the border layout manager.
import java.awt.*;
import javax.swing.*;

public class BorderPanel extends JPanel
{
   //  Sets up this panel with a button in each area of a border
   //  layout to show how it affects their position, shape, and size.
   public BorderPanel()
   {
      setLayout (new BorderLayout());

      setBackground (Color.green);

      JButton b1 = new JButton ("BUTTON 1");
      JButton b2 = new JButton ("BUTTON 2");
      JButton b3 = new JButton ("BUTTON 3");
      JButton b4 = new JButton ("BUTTON 4");
      JButton b5 = new JButton ("BUTTON 5");

      add (b1, BorderLayout.CENTER);
      add (b2, BorderLayout.NORTH);
      add (b3, BorderLayout.SOUTH);
      add (b4, BorderLayout.EAST);
      add (b5, BorderLayout.WEST);
   }
}
```





Outline

- GUI Elements
- More Components
- Layout Managers
- ⇨ Mouse and Key Events
- Dialog Boxes
- Important Details
- GUI Design

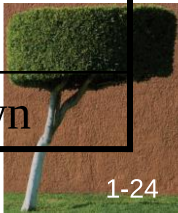


6.4 – Mouse and Key Events

- Events are also fired when a user interacts with the computer’s mouse and keyboard
- Mouse events

Mouse Event	**Description**
mouse pressed	The mouse button is pressed down
mouse released	The mouse button is released
mouse clicked	The mouse button is pressed down and released without moving the mouse in between
mouse entered	The mouse pointer is moved onto (over) a component
mouse exited	The mouse pointer is moved off of a component

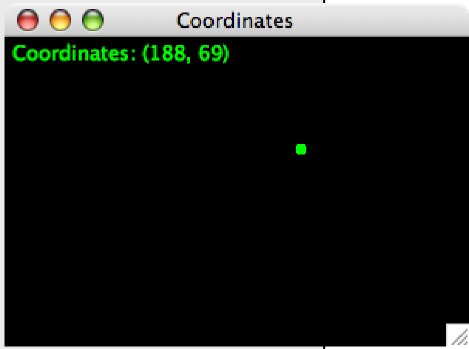
Mouse Motion Event	**Description**
mouse moved	The mouse is moved
mouse dragged	The mouse is moved while the mouse button is pressed down



6.4 – CoordinatesPanel.java

```
//  Represents the primary panel for the Coordinates program.
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class CoordinatesPanel extends JPanel
{
   private final int SIZE = 6;  // diameter of dot
   private int x = 50, y = 50;  // coordinates of mouse press

   //  Constructor: Sets up this panel to listen for mouse events.
   public CoordinatesPanel()
   {
      addMouseListener (new CoordinatesListener());

      setBackground (Color.black);
      setPreferredSize (new Dimension(300, 200));
   }
```

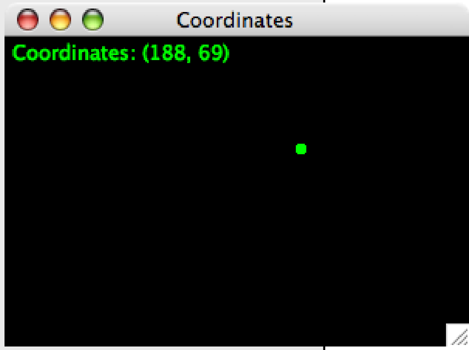
(more…)



Coordinates: (188, 69)

6.4 – CoordinatesPanel.java

```
   // Draws all of the dots stored in the list.
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);
      page.setColor (Color.green);
      page.fillOval (x, y, SIZE, SIZE);
      page.drawString ("Coordinates: (" + x + ", " + y + ")", 5, 15);
   }

   // Represents the listener for mouse events.
   private class CoordinatesListener implements MouseListener
   {
      //  Adds the current point to the list of points and redraws
      //  the panel whenever the mouse button is pressed.
      public void mousePressed (MouseEvent event)
      {
         x = event.getX();
         y = event.getY();
         repaint();
      }

      //  Provide empty definitions for unused event methods.
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
   }
}
```

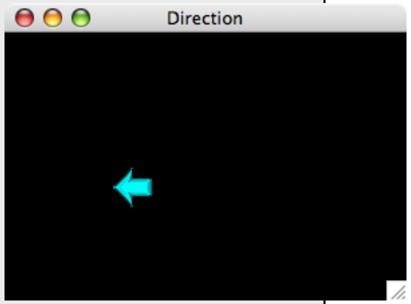


```
Coordinates: (188, 69)
```

6.4 – Mouse and Key Events

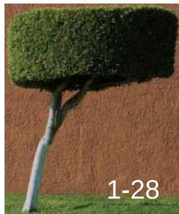
- Key Events
  - A *key event* is generated when the user presses a keyboard key

```
    private class DirectionListener implements KeyListener
    {
       public void keyPressed (KeyEvent event)
       {
          switch (event.getKeyCode())
          {
             case KeyEvent.VK_UP:
                currentImage = up;
                y -= JUMP;
                break;
             case KeyEvent.VK_DOWN:
                currentImage = down;
                y += JUMP;
                break;
             case KeyEvent.VK_LEFT:
                currentImage = left;
                x -= JUMP;
                break;
             …
          }
          repaint();
       }

       public void keyTyped (KeyEvent event) {}
       public void keyReleased (KeyEvent event) {}
    }
```



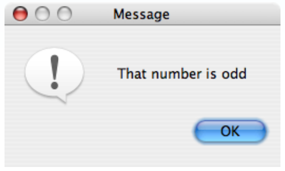
Outline

- GUI Elements
- More Components
- Layout Managers
- Mouse and Key Events
- ⇨ Dialog Boxes
- Important Details
- GUI Design

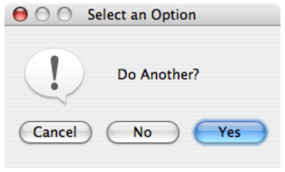


6.5 – Dialog Boxes

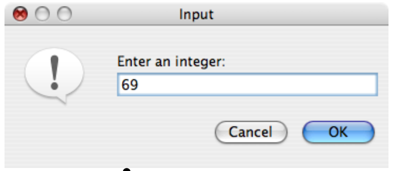
- A *dialog box* is a graphical window that pops up on top of any currently active window
- A dialog box can serve a variety of purposes
  - conveying information
  - confirming an action
  - permitting the user to enter information


Message

That number is odd


Select an Option

Do Another?


Input

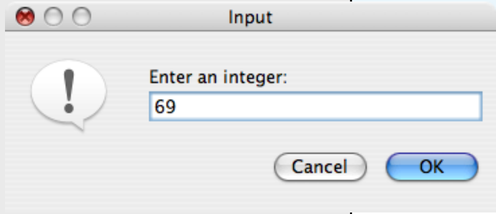
Enter an integer:

69

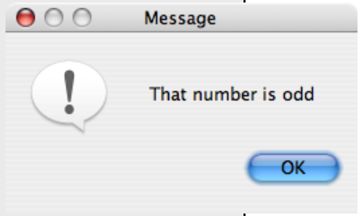
- `JOptionPane` dialog boxes fall into three categories
  - *message dialog boxes* – used to display an output string
  - *confirm dialog box* – presents the user with a simple yes-or-no question
  - *input dialog boxes* – presents a prompt and a single input txt file into which the user can enter one string of data

6.4 – EvenOdd.java

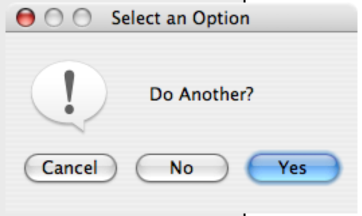
```
//  Demonstrates the use of the JOptionPane class.

import javax.swing.JOptionPane;

public class EvenOdd
{
   //  Determines if the value input by the user is even or odd.
   //  Uses multiple dialog boxes for user interaction.
   public static void main (String[] args)
   {
      String numStr, result;
      int num, again;

      do
      {
         numStr = JOptionPane.showInputDialog ("Enter an integer: ");
         num = Integer.parseInt(numStr);
         result = "That number is " + ((num%2 == 0) ? "even" : "odd");
         JOptionPane.showMessageDialog (null, result);
         again = JOptionPane.showConfirmDialog (null, "Do Another?");
      }
      while (again == JOptionPane.YES_OPTION);
   }
}
```



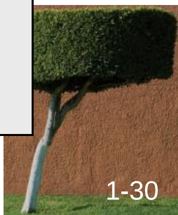
Enter an integer:
69



That number is odd

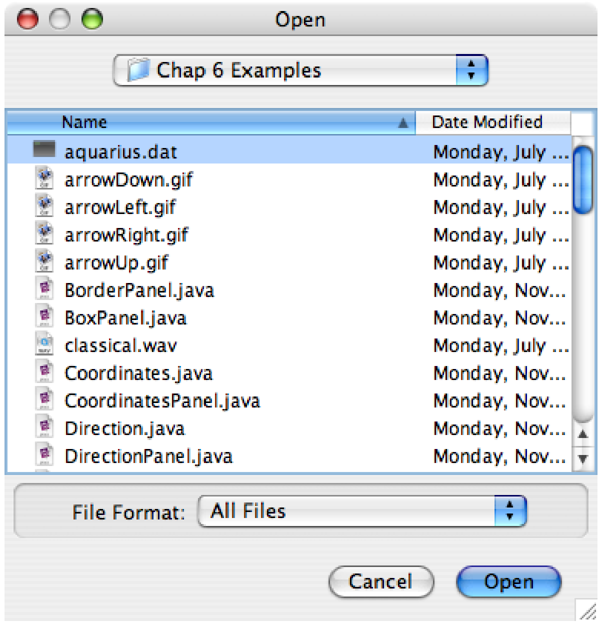


Do Another?



6.4 – Specialized Dialog Boxes

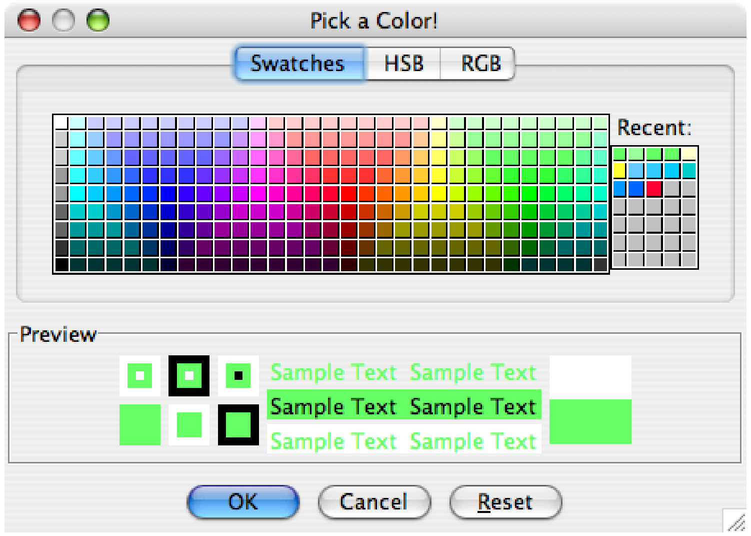
- A *file chooser* is a specialized dialog box used to select a file from a disk or other storage medium
- A *color chooser* dialog box can be displayed, permitting the user to select color from a list

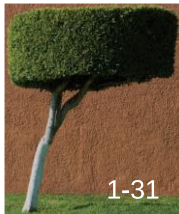

Open

Chap 6 Examples

Name	Date Modified
aquarius.dat	Monday, July ...
arrowDown.gif	Monday, July ...
arrowLeft.gif	Monday, July ...
arrowRight.gif	Monday, July ...
arrowUp.gif	Monday, July ...
BorderPanel.java	Monday, Nov...
BoxPanel.java	Monday, Nov...
classical.wav	Monday, July ...
Coordinates.java	Monday, Nov...
CoordinatesPanel.java	Monday, Nov...
Direction.java	Monday, Nov...
DirectionPanel.java	Monday, Nov...

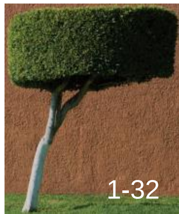
File Format: All Files





Outline

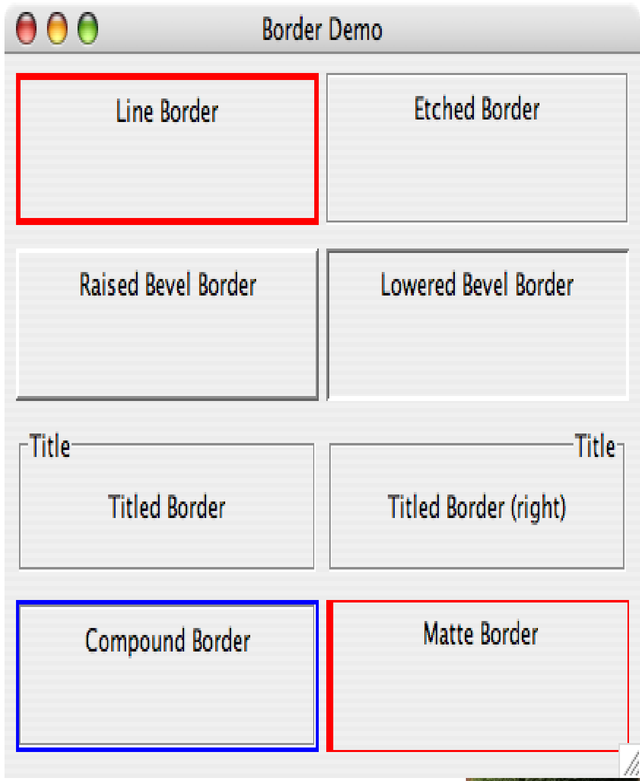
- GUI Elements
- More Components
- Layout Managers
- Mouse and Key Events
- Dialog Boxes
- ⇨ Important Details
- GUI Design

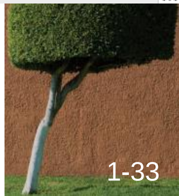


6.6 - Borders

- A border is not a component but defines how the edge of a component should be drawn

Border	Description
Empty Border	Puts a buffering space around the edge of a component, but otherwise has no visual effect
Line Border	A simple line surrounding the component
Etched Border	Creates the effect of an etched groove around a component
Bevel Border	Creates the effect of a component raised above the surface or sunken below it
Titled Border	Includes a text title on or around the border
Compound Border	A combination of two borders
Matte Border	Allows the size of each edge to be specified. Uses either a solid color or an image



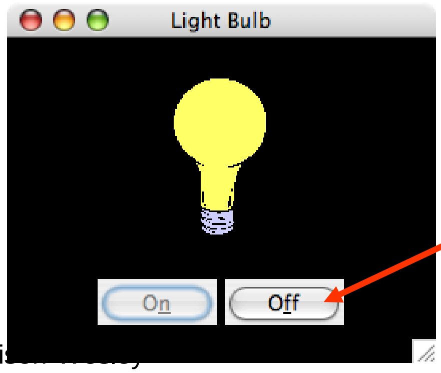


6.6 – Tool Tips and Mnemonics

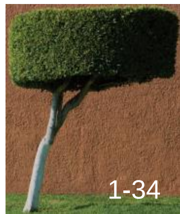
- A *tool tip* is a short line of text that appears over a component when the mouse cursor is rested momentarily on top of the component

```
JButton button = new Button (“Compute”);

button.setToolTipText (“Calculates the area under the curve”);
```

- A *mnemonic* is a character that allows the user to push a button or make a menu choice using the keyboard in addition to the mouse
  - We set the mnemonic for a component using the `setMnemonic` method of the component

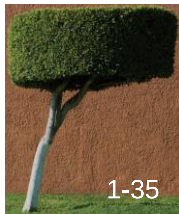


Mnemonic character ‘f’ set for the Off button



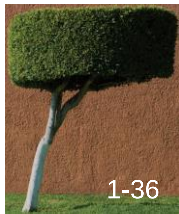
Outline

- GUI Elements
- More Components
- Layout Managers
- Mouse and Key Events
- Dialog Boxes
- Important Details
- ⇨ GUI Design



6.7 – GUI Design

- Keep in mind our goal is to solve a problem
- Fundamental ideas of good GUI design include
  - knowing the user
  - preventing user errors
  - optimizing user abilities
  - being consistent
- We should design interfaces so that the user can make as few mistakes as possible



Chapter 6 – Summary

- Chapter 6 focused on
  - discussing the core elements: components, events, and listeners
  - discussing the concept of a layout manager
  - examining mouse and keyboard events
  - describing tool tips and mnemonics
  - discussing GUI design issues