Window Interfaces Using Swing

Chapter 12

Introduction

- Modern programs don't simply read text input from the keyboard and write text output to the screen.
- Modern programs use window interfaces with buttons and menus, and allow users to make choices using a mouse.
- We will learn how to write simple window interfaces using the Swing library.

Introduction, cont.

- We'll also use an older library known as AWT (Abstract Windows Toolkit) which is a necessary complement to the Swing library.

GUIs - Graphical User Interfaces

- Windowing systems that interact with users often are called *GUIs*.
- A GUI accepts information from a user and makes it available to the program for processing.
- Most of the interaction is graphical in nature.

Definitions

- A *window* is a portion of the user's screen - a screen within a screen.
  - Typically, it has a border and a title.
- A *menu* is a list of alternatives available to the user.
  - A menu item is selected by using the mouse to place the cursor over the selected item, and by clicking the mouse.

Definitions, cont.

- A *button* serves much the same purpose as a menu item.
  - Typically, a button has a label.
  - The button is pushed by using the mouse to position the cursor over the button, and by clicking the mouse.

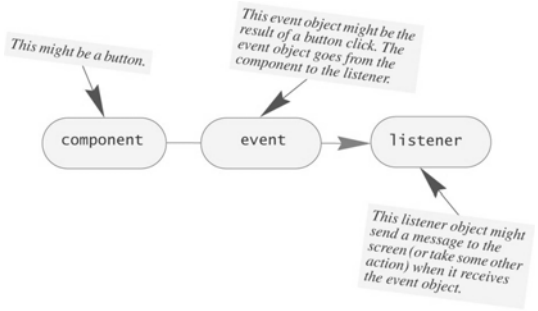
Event-Driven Programming

- Most GUI program use events and event handlers.
- A GUI *event* is an object that represents some action such as clicking a mouse, dragging a mouse, pressing a keyboard key, clicking the close-window button on a window, etc.
- When an object generates an event, it is said to *fire* the event.

Event-Driven Programming, cont.

- Objects that can fire events have one or more *listener* objects.
- The programmer chooses which event-firing objects have listeners.
- *Event handlers* are programmer-defined methods associated with the listener objects that determine what happens when events are detected by their associated listener(s).

Event-Driven Programming, cont.



Display 12.1
Event Firing and an Event Listener

Characteristics of Event-Driven Programming

- Our previous programs consisted of a list of statements to be executed in some order.
- In event-driven programming, events caused by user actions determine the upper-level order of activities.

Characteristics of Event-Driven Programming, cont.

- Each of us finds himself (or herself) in event-driven mode at one time or another.
  - We might be reading the newspaper waiting for the news to start.
  - But, if the phone rings, we answer it (or at least look at the caller ID).
  - If the doorbell rings, we answer it (or at least go to the door to see who is there).

Characteristics of Event-Driven Programming, cont.

  - If the news starts, we focus our attention on the TV.
  - If someone brings home pizza, we go to the kitchen to get a slice of pizza.
  - If the phone rings again, we answer the phone (or look once again at the caller ID).
  - etc.

Characteristics of Event-Driven Programming, cont.

- In general, it's impossible to predict this sequence of events in advance.
- When we write a GUI, there may be several methods that we never call directly.
- Instead, Swing calls these methods for us in response to events which have *listeners*.

Characteristics of Event-Driven Programming, cont.

- The classes that we define will be derived from classes in the Swing library.
- Sometimes we'll use inherited methods, and sometimes we'll override them.

Basic Swing Details

- We'll start with window elements to
  - close the window
  - put text in the window
  - color the window
  - put a title on the window.

Caution

- Sometimes, when running a Swing program, it becomes necessary to reboot the computer; any files that were left open may be damaged.
- Before running any Swing program that is not fully debugged, we should close all open files.

Example: A Simple Window

- This simple program produces a window and displays some text.
- If the close-window button is clicked, the program ends and the window disappears.
- The program is just a simple demo and it is not using the typical Swing style.

Example: A Simple Window, cont.

```
import javax.swing.*;
public class FirstSwingDemo
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    public static void main(String[] args)
    {
        JFrame myWindow = new JFrame( );
        myWindow.setSize(WIDTH, HEIGHT);
        JLabel myLabel =
                    new JLabel("Please don't click
  that button!");
        myWindow.getContentPane( ).add(myLabel);

... cont.
```

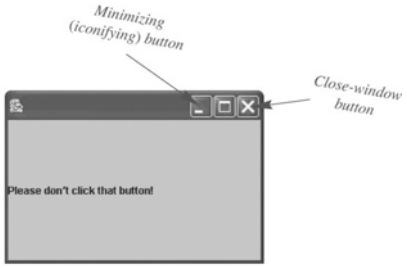
Example: A Simple Window, cont.

```
... cont.

        WindowDestroyer myListener = new
                        WindowDestroyer( );
        myWindow.addWindowListener(myListener);

        myWindow.setVisible(true);
    }
}
```

- Note: the class `WindowDestroyer` does not belong to Swing, we have to code it

Programming Example: A Simple Window, cont.

Resulting GUI



Display 12.2
A Very Simple Swing Demonstration Program

Size Units for Screen Objects

- The size of an object on the screen is measured in pixels.
- A *pixel* is the smallest unit of screen space onto which you can write.
- Pixels do not represent fixed lengths, but depend instead on the size and resolution of the screen.
- The exact size of what is produced varies from screen to screen.

More on `setVisible`

- Method `setVisible` takes one argument of type `boolean`.
- Example

```
w.setVisible(true);
```

- Method `setVisible` permits the programmer to specify when GUI objects should be displayed and when they should not.

Window Listeners

- A *window listener* listens to events from a window, such as a click on the close-window button.
- A window listener is *registered* when it becomes associated with the object(s) (i.e., calling `addWindowListener`) to which it listens.

Class `WindowDestroyer`

- A listener class often is derived from class `WindowAdapter`.
- The inherited methods respond automatically to different kinds of events.
- The way an event is handled depends on the programmer.
- Typically, an inherited method definition is overridden.

Class `WindowDestroyer`, cont.

```
import java.awt.*;
import java.awt.event.*;
/* Registering an object of this class as a listener to
   any object of the class JFrame, then if the user
   clicks the close-window button in the JFrame, the
   object of this class will end the program and close
   the JFrame.*/
public class WindowDestroyer extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```


Ending a Swing Program

- A GUI program is based on a kind of infinite loop.
- The windowing system normally stays on the screen until the user indicates that it should go away.
- The `exit` method ends a Java program as soon as the `exit` method is executed.

```
System.exit(0);
```

Ending a Swing Program, cont.

- If the window-close button is not programmed (i.e. `windowClosing`), a click causes the window to disappear, but does not cause the program to end.

Methods of Class `WindowAdapter`

`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.

`public void windowIconified(WindowEvent e)`
Invoked when a window is iconified. When you click the minimize button in a `JFrame`, the window is iconified. See Display 12.2 for the location of the minimizing (iconifying) button.

Methods of Class `WindowAdapter`, cont.

`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 any window is activated, all other windows are deactivated. Other actions can also deactivate a window.

`public void windowGainedFocus(WindowEvent e)`
Invoked when a window gains focus. (Focus is not discussed in this text.)

`public void windowLostFocus(WindowEvent e)`
Invoked when a window loses focus. (Focus is not discussed in this text.)

`public void windowStateChanged(WindowEvent e)`
Invoked when a window changes state.

Methods of Class `WindowAdapter`, cont.

- Because class `WindowAdapter` is abstract, this class can be used only as a base class for defining other classes.
- When you define a derived class of abstract class `WindowAdapter`, you override and redefine only those methods that you need.

Programming Example: Improved Swing Program

- This program presents two windows rather than just one.
- Further, this program demonstrates an appropriate style for writing GUIs.
  - The definition of the window is in a separate class, derived from class `JFrame`.
  - The window is displayed in a program that uses the class.

Improved Swing Program, cont.

```
import javax.swing.*;
public class FirstWindow extends JFrame {
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public FirstWindow( ){
        super( );
        setSize(WIDTH, HEIGHT);
        JLabel myLabel = new JLabel("Please don't
  click that button!");
        getContentPane( ).add(myLabel);
        WindowDestroyer listener = new
                              WindowDestroyer( );
        addWindowListener(listener);
    }}
```

Improved Swing Program, cont.

```
import javax.swing.*;

public class FirstWindowDemo
{
    public static void main(String[] args)
    {
        FirstWindow window1 = new FirstWindow( );
        window1.setVisible(true);

        FirstWindow window2 = new FirstWindow( );
        window2.setVisible(true);
    }
}
```


Improved Swing Program, cont.

Resulting GUI

If it looks as though you have only one window when you run this program, move the window. The windows may be one on top of the other.

Display 12.7
A Program That uses the Class FirstWindow

Adding Items to a `JFrame`

- Inside a constructor a `JLabel` can be added to a `JFrame` using

```
getContentPane().add(Jlabel_Name);
```

- Note: don't use the method `add` directly!!

Programming Example: A Window with Color

- We add four new features:
  - a title
  - a background color
  - a local variable named contentPane
  - a new way to add the window listener

Example: A Window with Color

```
import javax.swing.*;
import java.awt.*; //needed for the Color class
public class SecondWindow extends JFrame{
    public static final int WIDTH = 200;
    public static final int HEIGHT = 200;
    public SecondWindow( ){
        super( );
        setSize(WIDTH, HEIGHT);
        Container contentPane = getContentPane( );
        JLabel label = new JLabel("Now available in
color!");
        contentPane.add(label);
        setTitle("Second Window");
        contentPane.setBackground(Color.BLUE);
        addWindowListener(new WindowDestroyer( ));
}   cont.
```


Example: A Window with Color – cont.

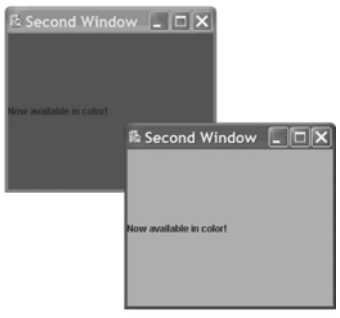
```
public SecondWindow(Color customColor)
    {
        super( );
        setSize(WIDTH, HEIGHT);

        Container contentPane = getContentPane( );
        JLabel label = new JLabel("Now available in
color!");
        contentPane.add(label);

        setTitle("Second Window");
        contentPane.setBackground(customColor);

        addWindowListener(new WindowDestroyer( ));
   }
}
```

Example: A Window with Color – cont.

```
import java.awt.*; //for the class Color
public class SecondWindowDemo
{   /*
    Creates and displays two windows
    of the class SecondWindow.
    */
    public static void main(String[] args)
    {
        SecondWindow window1 = new SecondWindow( );
        window1.setVisible(true);

        SecondWindow window2 = new
                          SecondWindow(Color.PINK);
        window2.setVisible(true);
    }
}
```


Programming Example: A Window with Color, cont.

Resulting GUI



Display 12.10
A Demonstration Program for SecondWindow

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
```

The class `Color` is in the AWT package (library). So when using these colors, you need the following `import` statement:

```
import java.awt.*;
```

Display 12.9
The Color Constants

Anonymous Objects

- Some methods (`addWindowListener`, for example) need objects as arguments, even when no subsequent reference to such objects is needed.
- Such an object can be created and passed without naming it. Example

```
addWindowListener( new
Windowdestroyer());
```

Some Methods of Class `JFrame`

Method	Description
`JFrame()`	Constructor for creating a new `JFrame`.
`JFrame(String title)`	Constructor for creating a new `JFrame` with the specified `title`.
`add`	`JFrame` has a method `add`, but it should not be used. (It is basically a useless inheritance from an ancestor class). To add something to a `JFrame`, use `getContentPane().add(`*Item_Added*`)`
`void addWindowListener( WindowListener ear)`	Registers `ear` as a listener for events fired by the `JFrame`.
`Container getContentPane()`	Returns the content-pane object of the `JFrame`. Note that the content pane that is returned is of type `Container`.
`void setBackground(Color c)`	Sets the background color to `c`.
`void setForeground(Color c)`	Sets the foreground color to `c`.
`void setSize(int width, int height)`	Resizes the window to the specified width and height.
`void setTitle(String title)`	Displays the `title` on the title bar of the window.
`void setVisible(boolean b)`	Makes the window visible if the argument is `true`. Makes it invisible if the argument is `false`.

Display 12.11
Some Methods in the Class J Frame

Some Methods of Class `JFrame`, cont.

- A window class normally is derived from class `JFrame`.
- A derived window class inherits all the methods (described on the previous slide(s)) from class `JFrame`.

What to Import

- It may be simpler to use

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

  rather than trying to determine which import statements are needed for a particular window interface.

  - `event.*` represents a package within `java.awt`.

Layout Managers

- The objects that you add to a container class are arranged by an object known as a *layout manager.*
- A layout manager is added using method `setLayout`, which is a method of every container class.
- syntax

```
Container_Object.setLayout(new
  Layout_Manager_Class(Any_Parameters));
```

Border Layout

- Border layout can be used to arrange three items vertically instead of horizontally.
- `class BorderLayoutDemo`

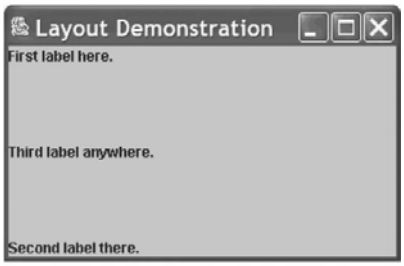
```
import javax.swing.*;
import java.awt.*;

/**
 Simple demonstration of the use of a layout manager
 to arrange labels.
*/
public class BorderLayoutDemo extends JFrame
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    /**
     Creates and displays a window of the class BorderLayoutDemo.
    */
    public static void main(String[] args)
    {
        BorderLayoutDemo gui = new BorderLayoutDemo();
        gui.setVisible(true);
    }
```

```
    public BorderLayoutDemo()
    {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Layout Demonstration");
        Container content = getContentPane();

        content.setLayout(new BorderLayout());

        JLabel label1 = new JLabel("First label here.");
        content.add(label1, BorderLayout.NORTH);

        JLabel label2 = new JLabel("Second label there.");
        content.add(label2, BorderLayout.SOUTH);

        JLabel label3 = new JLabel("Third label anywhere.");
        content.add(label3, BorderLayout.CENTER);
    }
}
```

Display 12.12
Using the Border Layout Manager

Border Layout, cont.

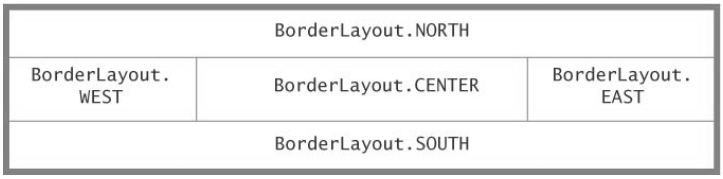
Resulting GUI



Display 12.12
Using the BorderLayout Manager

Border Layout, cont.

- A BorderLayout manager can place a component into any of five regions.
- Regions which are unused give up their space to `BorderLayout.CENTER`.



Border Layout, cont.

- equivalent forms:

```
content.add(label3,
    BorderLayout.CENTER);
```

and

```
content.add(label3, "Center");
```

Flow Layout

- The simplest layout manager is the `FlowLayout` manager.
- Components are added and arranged one after another, left to right, until a row is filled. Then components are added to the next row in the same manner.
- Each row is centered in its container.

Grid Layout

- A `GridLayout` manager arranges components in a grid of rows and columns.
- Example

```
aContainer.setLayout(new
    GridLayout(2,3));
```

Grid Layout, cont.

- Each entry has the same size.
- Rows are filled one at a time, top to bottom, and from left to right within each row.
- Even though the number of columns is specified, the actual number of columns is determined by the number of items added to the container.

Some Layout Managers

Layout Manager	Description
`FlowLayout`	Displays components from left to right in the same fashion that you normally write things on a piece of paper.
`BorderLayout`	Displays the components in five areas: north, south, east, west, and center. You specify which area a component goes into in a second argument of the `add` method.
`GridLayout`	Lays components out in a grid, with each component stretched to fill its box in the grid.

Display 12.13
Some Layout Managers

Default Layout Managers

- When a default manager is not added explicitly, a default layout manager is provided.
- The default manager for the content pane of a `JFrame` is `BorderLayout`.
- The default manager for a `JPanel` is `FlowLayout`.

Buttons

- A *button* is a GUI component that looks like a button and does something when it is clicked using a mouse.
- Like a label, a button is created and added to a container.
- Unlike a label, a button can fire an event and the event can cause a GUI to perform some action.

Adding Buttons

- A button is created using

```
JButton Button_Name = new
    JButton("Button_Label");
```

- A button is added to a container using

```
Container_Name.add(Button_Name);
```


Adding Buttons, cont.

- `class ButtonDemo`

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

/**
 Simple demonstration of putting buttons in a JFrame.
*/
public class ButtonDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    /**
     Creates and displays a window of the class ButtonDemo.
    */
    public static void main(String[] args)
    {
        ButtonDemo buttonGui = new ButtonDemo();
        buttonGui.setVisible(true);
    }

    public ButtonDemo()
    {
        setSize(WIDTH, HEIGHT);

        addWindowListener(new WindowDestroyer());
        setTitle("Button Demo");
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.BLUE);
        contentPane.setLayout(new FlowLayout());

        JButton stopButton = new JButton("Red");
        stopButton.addActionListener(this);
        contentPane.add(stopButton);

        JButton goButton = new JButton("Green");
        goButton.addActionListener(this);
        contentPane.add(goButton);
    }

    It will take more than one subsection to fully explain this
    program. The explanation does not end until the end of the
    subsection entitled "Action Listeners and Action Events."

    public void actionPerformed(ActionEvent e)
    {
        Container contentPane = getContentPane();

        if (e.getActionCommand().equals("Red"))
            contentPane.setBackground(Color.RED);
        else if (e.getActionCommand().equals("Green"))
            contentPane.setBackground(Color.GREEN);
        else
            System.out.println("Error in button interface.");
    }
}
```

Display 12.14
A GUI with Buttons Added

Adding Buttons, cont.

Resulting GUI

Display 12.14
A GUI with Buttons Added

Close-Window Buttons and `JButtons`

- A button added to a GUI is an object of class `JButton`.
- A close-window button is not an object of class `JButton`. Instead, it is part of a `JFrame` object.

Action Listeners and Action Events

- For each button, the GUI needs to
  - register (specify) the listener object(s).
  - define the methods to be invoked when an event is fired.
- For a statement such as

```
stopButton.addActionListener(this);
```

the class `ButtonDemo` is itself the listener class.

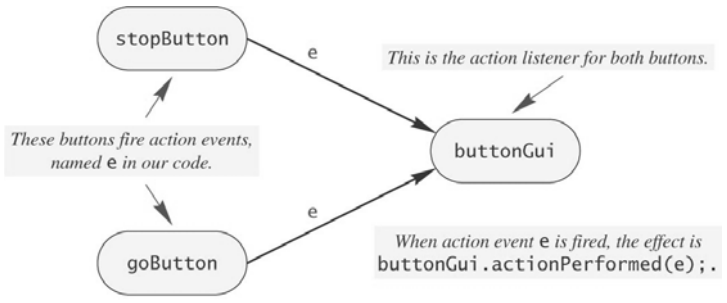
Action Listeners and Action Events, cont.

- Different kinds of components requite different kinds of listener classes.
- Buttons fire ***action events*** which are handled by *action listeners*.
- An action listener is an object of type `ActionListener`, and `ActionListener` is an *interface*.
- Any class can be an `ActionListener` class…

Action Listeners and Action Events, cont.

- To make a class into an `ActionListener`
  - add `implements ActionListener` to the heading of the class definition
  - define a method named `ActionPerformed`.
  - register the `ActionListener` object with the component that will fire the event using the method `addActionListener`
  - (A component may register with more than one listener.)

Buttons and an Action Listener



Display 12.15
Buttons and an Action Listener

The `actionPerformed` Method

- An `ActionListener` class must have a method named `actionPerformed` that has one parameter of type `ActionEvent`.
- syntax

```
public void actionPerformed(ActionEvent e)
{
    Code_for_Actions_Performed
}
```


Interfaces

- An *interface* is a property of a class that states what methods the class must define.
- `ActionListener` is an interface.
- A class which satisfies the requirements of an interface *implements the interface*.
- A class can define methods in addition to the methods required by the interface.
- An interface is not a class, but it is a type.

Interfaces, cont.

- To implement an interface, a class must
  - include the phrase `implements` *`Interface_Name`* at the start of the class definition
  - implement all the method headings listed in the definition of the interface.
- A programmer can define his own interfaces.

Multiple Interfaces

- A class which implements multiple interfaces lists the names of all the interfaces, separated by commas.

```
implements First_Interface_Name,
    Second_Interface_Name, …,
    Last_Interface_Name
```

- The class must implement all the methods of all the listed interfaces.

Code a GUIs Appearance and Actions Separately

- Code for a Swing GUI is simpler if it is divided into two parts:
  - the GUI's appearance on the screen
  - the GUI's actions.
- In a complicated Swing GUI, either of these tasks by itself can be formidable.

Method `setActionCommand`

- We can think of the method invocation `e.getActionCommand()` as returning the string written on the button.
- In fact, this method invocation returns a string known as the *action command* for the button.
- A different action command can be specified for the button using the `setActionCommand`

Method `setActionCommand`, cont.

- example

```
JButton stopButton =
    new JButton("Red");
stopButton.setActionCommand("Stop");
```

- This permits the same string to be written on two different buttons, but with the two buttons distinguished from one another by the program.

Method `setActionCommand`, cont.

- Every object that fires an action event has an associated string known as the *action command* for that component.
- `e.getActionCommand()` returns the action command for the component that fired `e`.
- The default action command for a button is the string written on it.
- Method `setActionCommand` can be used to change the action command for the object.

The `JPanel` Class

- A GUI can be organized hierarchically, with window-like containers inside other window-like containers.
- Class `JPanel` is a simple container that does little more than hold components.
- Components can be placed in a `JPanel` which can be placed in another `JPanel`, … which can be placed in a `JFrame` (better, in its `ContentPane`).

The `JPanel` Class, cont.

- To place two components (e.g., 2 buttons) in `BorderLayout.SOUTH` for example, simple place the two components in a panel and place the panel in the `BorderLayout.SOUTH` position.
- The panel has its own layout manager.

The JPanel Class, cont.

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

/**
 Simple demonstration of putting buttons in a panel.
*/
public class PanelDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    public static void main(String[] args)
    {
        PanelDemo guiWithPanel = new PanelDemo();
        guiWithPanel.setVisible(true);
    }

    public PanelDemo()
    {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Panel Demonstration");
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.BLUE);
        contentPane.setLayout(new BorderLayout());

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

        JButton stopButton = new JButton("Red");
        stopButton.setBackground(Color.RED);
        stopButton.addActionListener(this);
        buttonPanel.add(stopButton);

        JButton goButton = new JButton("Green");
        goButton.setBackground(Color.GREEN);
        goButton.addActionListener(this);
        buttonPanel.add(goButton);

        contentPane.add(buttonPanel, BorderLayout.SOUTH);
    }
    public void actionPerformed(ActionEvent e)
    {
       Container contentPane = getContentPane();

       if (e.getActionCommand().equals("Red"))
            contentPane.setBackground(Color.RED);
       else if (e.getActionCommand().equals("Green"))
          contentPane.setBackground(Color.GREEN);
       else
            System.out.println("Error in button interface.");
    }
}
```

Display 12.17
Putting the Buttons in a Panel

The JPanel Class, cont.

Resulting GUI

Display 12.17
Putting the Buttons in a Panel

The `Container` Class, cont.

- The hierarchy of Swing classes (vedere libro di testo pag. 879)

Abstract Classes

- An *abstract class* is a placeholder in a hierarchy typically and is used to consolidate characteristics of all of its descendants.
- You cannot instantiate objects of an abstract class directly.
- An abstract class can serve as the base class for derived classes which can be instantiated.

The `Container` Class

- Class `Container` is a predefined class.
- An object of a class which descends from class `Container` is called a *container class* and can have components added to it.
- Class `JFrame` is a descendent of class `Container`, permitting any `JFrame` object to hold labels, buttons, panels, and other components.

The `Container` Class, cont.

- Class `JPanel` is a descendent of class `Container`, permitting any `JPanel` object to hold labels, buttons, other panels, and other components.

The JComponent Class

- Class `JComponent` is similar to class `Container`, and plays a similar role for components.
- Any class that descends from `JComponent` is called a `JComponent` class.
- Any `JComponent` object can be added to any `Container` object.

The JComponent Class, cont.

- Since class `JComponent` descends from class `Container`, a `JComponent` object can be added to another `JComponent` object.

Adding Components

- To add a component to a `JFrame`, use method `getContentPane` to obtain the content pane, and the use method `add` with the content pane as the calling object.
- example

```
Container contentPane = getContentPane();
Jlabel label = new Jlabel("Click Here");
contentPane.add(label);
```


Adding Components, cont.

- For other container classes, add components by using method `add` directly with an object of the container class.
- example

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

Objects in Swing Containers

- Swing containers use three kinds of objects:
  - the container class itself (such as a panel)
  - the components (labels, buttons, panels, etc.)
  - the layout manager.
- Typically, a GUI interface, and many subparts of the GUI, will consist of these three kinds of objects.

Creating Simple Window Interfaces: Guidelines

- A typical GUI consists of a windowing object derived from class `JFrame`, together with components such as labels and buttons.
- The programmer must register a window listener to close the window.
- Components are grouped by placing them in a `JPanel` and by adding the `JPanel` to the GUI.

Creating Simple Window Interfaces: Guidelines, cont.

- The GUI (i.e. the `JFrame`) and each `JPanel` should be given a layout manager.
- The GUI (or some other class) needs to be made an action listener, by implementing `ActionListener`, for components which generate events of interest.
- Each such component should have an action listener registered with it, and an appropriate `actionPerformed` method defined.

Text I/O for GUIs: Outline

- Text Areas and Text Fields
- Inputting and Outputting Numbers
- Catching a `NumberFormatException`

Text Areas and Text Fields, cont.

Resulting GUI

Display 12.19
A GUI with a Text Area

Text Areas and Text Fields

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

public class MemoSaver extends JFrame implements ActionListener
{
    public static final int WIDTH = 600;
    public static final int HEIGHT = 300;
    public static final int LINES = 10;
    public static final int CHAR_PER_LINE = 40;

    private JTextArea theText;
    private String memo1 = "No Memo 1.";
    private String memo2 = "No Memo 2.";

    public MemoSaver()
    {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Memo Saver");
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.setLayout(new FlowLayout());
        JButton memo1Button = new JButton("Save Memo 1");
        memo1Button.addActionListener(this);
        buttonPanel.add(memo1Button);
        JButton memo2Button = new JButton("Save Memo 2");
        memo2Button.addActionListener(this);
        buttonPanel.add(memo2Button);
        JButton clearButton = new JButton("Clear");
        clearButton.addActionListener(this);
        buttonPanel.add(clearButton);
        JButton get1Button = new JButton("Get Memo 1");
        get1Button.addActionListener(this);
        buttonPanel.add(get1Button);
        JButton get2Button = new JButton("Get Memo 2");
        get2Button.addActionListener(this);
        buttonPanel.add(get2Button);
        contentPane.add(buttonPanel, BorderLayout.SOUTH);
        JPanel textPanel = new JPanel();
        textPanel.setBackground(Color.BLUE);
```

There is a demonstration `main` *in part 2 of this display.*

If you get `memo 1` *before you set* `memo 1`*, you get the message* `"No Memo 1."`

Display 12.19
A GUI with a Text Area

Text Areas and Text Fields, cont.

```
        theText = new JTextArea(LINES, CHAR_PER_LINE);
        theText.setBackground(Color.WHITE);
        textPanel.add(theText);
        contentPane.add(textPanel, BorderLayout.CENTER);
    }

    public void actionPerformed(ActionEvent e)
    {
        String actionCommand = e.getActionCommand();
        if (actionCommand.equals("Save Memo 1"))
            memo1 = theText.getText();
        else if (actionCommand.equals("Save Memo 2"))
            memo2 = theText.getText();
        else if (actionCommand.equals("Clear"))
            theText.setText("");
        else if (actionCommand.equals("Get Memo 1"))
            theText.setText(memo1);
        else if (actionCommand.equals("Get Memo 2"))
            theText.setText(memo2);
        else
            theText.setText("Error in memo interface");
    }

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

theText is an instance variable.

Display 12.19
A GUI with a Text Area

Text Areas and Text Fields, cont.

- Method `getText` returns the text written in an object of class `JTextArea`.
- Method `setText` of class `JTextArea` changes the text in the text area into whatever is provided as the argument to method `setText`.
- Class `JTextField` is similar to class `JTextArea`, but displays only one line of text.

Text Areas and Text Fields, cont.

- Classes `JTextArea` and `JTextField` have a constructor with no arguments; it sets the parameters to default values.
- Both classes can have some text specified when `new` is used to create an object.
- example

```
JTextField IOField =
     new JTextField("Aloha!", 20);
```


Text Areas and Text Fields, cont.

- An object of class `JTextArea` can have a size consisting of a specified number of lines and a specified number of characters per line.
- example

```
JTextArea someText =
    new JTextArea(10,30);
```

Text Areas and Text Fields, cont.

- An object of class `JTextField` can have a size consisting of a specified number of characters.
- example

```
JTextField name =
    new JTextField(10);
```


Text Areas and Text Fields, cont.

- The number of characters (per line) is not absolute, but represents the space needed for one `'m'` character.

Line Wrapping in Text Areas

- Method `setLineWrap` sets the line wrapping policy for a `JTextArea` object.
- example
```
theText.setLineWrap(true);
```
- If the argument is set to `false`, extra characters will be on the same line, but will not be visible.

Read-Only Text Components

- To specify that a user cannot write in a `JTextArea` or a `JTextField`, use method `setEditable`.
- example
```
theText.setEditable(false);
```
- A `JTextArea` or a `JTextField` can be made editable subsequently using, for example
```
theText.setEditable(true);
```

Programming Example: Labeling a Text Field

- Typically, a label precedes a text field to tell the user what information is needed in the text field.

Labeling a Text Field, cont.

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

/**
 Class to demonstrate placing a label on a text field.
*/
public class LabelDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    private JTextField name;

    public LabelDemo()
    {
        setTitle("Name Tester");
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        Container content = getContentPane();
        content.setLayout(new GridLayout(2, 1));

        JPanel namePanel = new JPanel();
        namePanel.setLayout(new BorderLayout());
        namePanel.setBackground(Color.LIGHT_GRAY);

        name = new JTextField(20);
        namePanel.add(name, BorderLayout.SOUTH);
        JLabel nameLabel = new JLabel("Enter your name here:");
        namePanel.add(nameLabel, BorderLayout.CENTER);

        content.add(namePanel);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());
        JButton b = new JButton("Test");
        b.addActionListener(this);
        buttonPanel.add(b);
        b = new JButton("Clear");
        b.addActionListener(this);
        buttonPanel.add(b);

        content.add(buttonPanel);
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getActionCommand().equals("Test"))
            name.setText("A very good name!");
        else if (e.getActionCommand().equals("Clear"))
            name.setText("");
        else
            name.setText("Error in window interface.");
    }

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

Nota: usa 3 layout managers differenti

Programming Example: Labeling a Text Field, cont.

Resulting GUI

Display 12.20
Labelling a Text Field

Inputting and Outputting Numbers

- Input provided using a `JTextArea` object or `JTextField` object is received as a string.
- When numeric input is needed, the string must be converted to a number.
- To output a number using a GUI constructed with Swing, the number must be converted to a string.
- All input typed by the user is string input, and all displayed output is string output.

Inputting and Outputting Numbers, cont.

- To convert a string to an integer, use, for example

```
Integer.parseInt("42");
```

or

```
Integer.parseInt(ioField.getText());
```

or, to eliminate whitespace before or after the input, use

```
Integer.parseInt
    (ioField.getText().trim());
```


Inputting and Outputting Numbers, cont.

- To input numbers of type double, use

```
Double.parseDouble(ioField.getText().trim());
```

- Analogous conversions can be done with classes `Long` and `Float`.

Inputting and Outputting Numbers, cont.

- Code can be made simpler by defining a method such as

```
private static int stringToInt(String s)
{
    return Integer.parseInt(s.trim());
}
```

and then using

```
n = stringToInt(ioField.getText());
```


Inputting and Outputting Numbers, cont.

- To write numeric output to a `JTextArea` or a `JTextField`, use method `toString`.
- examples

```
Integer.toString(sum);
Double.toString(average);
ioField.setText(Integer.toString(sum));
```

A GUI Adding Machine

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

/**
 GUI for totaling a series of numbers.
*/
public class Adder extends JFrame implements ActionListener

    public static final int WIDTH = 400;
    public static final int HEIGHT = 200;

    private JTextField inputOutputField;
    private double sum = 0;

    public static void main(String[] args)
    {
        Adder guiAdder = new Adder();
        guiAdder.setVisible(true);
    }

    public Adder()
    {
        setTitle("Adding Machine");
        addWindowListener(new WindowDestroyer());
        setSize(WIDTH, HEIGHT);
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.GRAY);
        buttonPanel.setLayout(new FlowLayout());
        JButton addButton = new JButton("Add");
        addButton.addActionListener(this);
        buttonPanel.add(addButton);
        JButton resetButton = new JButton("Reset");
        resetButton.addActionListener(this);
        buttonPanel.add(resetButton);
        contentPane.add(buttonPanel, BorderLayout.SOUTH);

        JPanel textPanel = new JPanel();
        textPanel.setBackground(Color.BLUE);
        textPanel.setLayout(new FlowLayout());

          inputOutputField = new JTextField("Numbers go here.", 30);
          inputOutputField.setBackground(Color.WHITE);
          textPanel.add(inputOutputField);
          contentPane.add(textPanel, BorderLayout.CENTER);
    }
```

Display 12.21
An Addition GUI

A GUI Adding Machine, cont.

```
    public void actionPerformed(ActionEvent e)
    {
        if (e.getActionCommand().equals("Add"))
        {
            sum = sum +
                stringToDouble(inputOutputField.getText());
            inputOutputField.setText(Double.toString(sum));
        }
        else if (e.getActionCommand().equals("Reset"))
        {
            sum = 0;
            inputOutputField.setText("0.0");
        }
        else
            inputOutputField.setText("Error in adder code.");
     }

    private static double stringToDouble(String stringObject)
    {
        return Double.parseDouble(stringObject.trim());
    }
}
```

Display 12.21
An Addition GUI

Programming Example: A GUI Adding Machine, cont.

Resulting GUI

Display 12.21
An Addition GUI

Catching a `NumberFormatException`

- A GUI, such as class `Adder`, has no control over what the user enters in the text field. The user might enter commas, or even alphabetic characters, resulting an a `NumberFormatException`, which leaves the GUI in an unpredictable state.

Catching a `NumberFormatException`, cont.

- A `NumberFormatException` can be caught, and the user can be asked to reenter the number.

Catching a `NumberFormatException`, cont.

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

/**
 GUI for totaling a series of numbers. If the user
 enters a number in an incorrect format, such as
 2,000 with a comma, then an error message is generated
 and the user can restart the computation.
*/
public class ImprovedAdder extends JFrame
                           implements ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 200;

    private JTextField inputOutputField;
    private double sum = 0;

    public static void main(String[] args)
    {
        ImprovedAdder guiAdder = new ImprovedAdder();
        guiAdder.setVisible(true);
    }

    public ImprovedAdder()
        <The rest of the definition is the same as the constructor
                        Adder in Display 12.21 >

    public void actionPerformed(ActionEvent e)
    {
        try
        {
            tryingCorrectNumberFormats(e);
        }
        catch (NumberFormatException e2)
        {
            inputOutputField.setText("Error: Reenter Number.");
        }
    }
```

This class is identical to the class `Adder` *in Display 12.21, except that the name of the class is changed and the method* `actionPerformed` *is changed.*

`NumberFormatException`*s do not need to be declared in a* `throws` *clause, but they can be caught like other exceptions.*

```
    //This method can throw a NumberFormatException.
    public void tryingCorrectNumberFormats(ActionEvent e)
    {
        if (e.getActionCommand().equals("Add"))
        {
            sum = sum +
                  stringToDouble(inputOutputField.getText());
            inputOutputField.setText(Double.toString(sum));
        }
        else if (e.getActionCommand().equals("Reset"))
        {
            sum = 0;
            inputOutputField.setText("0.0");
        }
        else
            inputOutputField.setText("Error in adder code.");
    }
    //This method can throw a NumberFormatException.
    private static double stringToDouble(String stringObject)
    {
        return Double.parseDouble(stringObject.trim());
    }
}
```

Display 12.22
A GUI with Exception Handling

Catching a `NumberFormatException`, cont.

Resulting GUI

As long as the user enters correctly formatted numbers, this GUI behaves exactly the same as the one in Display 12.21.

If the user enters a number in an incorrect format, such as a number with a comma, the GUI looks like this.

Display 12.22
A GUI with Exception Handling

Summary

- You have learned the basics of event-driven programming.
- You have designed and coded a simple GUI with buttons and text.
- You have learned about several Swing-related classes.