Chapter 12

Window Interfaces Using Swing Objects

- Event-Driven Programming and GUIs
- Swing Basics and a Simple Demo Program
- Layout Managers
- Buttons and Action Listeners
- Container Classes
- Text I/O for GUIs
- Inner Classes

Swing

- Special library of classes that allows Java programs to have a windowing interface
- Part of larger collection called *Java Foundation Classes* or *JFC*
- An improved version of older library called *Abstract Window Toolkit* (*AWT*)
- Standard part of all versions of Java 2 (JDK 1.2)

GUIs—Graphical User Interfaces

Most modern programs use a GUI

GUI (pronounced “gooey”):

- **G**raphical—not just text or characters: windows, menus, buttons, etc.
- **U**ser—person using the program
- **I**nterface—way to interact with the program

Typical graphical elements:

- *Window*—portion of screen that serves as a smaller screen within the screen
- *Menu*—list of alternatives offered to user
- *Button*—looks like a button that can be pressed

Event-Driven Programming

- Programs with GUIs often use *Event-Driven Programming*
- Program waits for events to occur and then responds
- Examples of events:
  - » Clicking a mouse button
  - » Dragging the mouse
  - » Pressing a key on the keyboard
- *Firing an event*—when an object generates an event
- *Listener*—object that waits for events to occur
- *Event handler*—method that responds to an event

A New Approach to Programming

Previous Style of Programming:

- List of instructions performed in order
- Next thing to happen is next thing in list
- Program performed by one agent—the computer

Event-Driven Style of Programming:

- Objects that can fire events and objects that react to events
- Next thing to happen depends on next event
- Program is interaction between user and computer

Very Simple Swing Demonstration

```
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…");
    myWindow.getContentPanel().add(myLabel);

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

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

Notes on the Simple Demo Program

```
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…");
    myWindow.getContentPanel().add(myLabel);

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

    myWindow.setV
  }
}
```

Used in all Swing programs

Creates a `JFrame` window named `myWindow`

Adds a label to the `JFrame` window—note use of `getContentPane`

Notes on the Simple Demo Program

```
import javax.swing.*;
public class FirstSwingDemo
{
  public static final int WIDTH = 3
  public static final int HEIGHT =

  public static void main(String[]

    JFrame myWindow = new JFrame();
    myWindow.setSize(WIDTH, HEIGHT);
    JLabel myLabel = new JLabel("Pleas don't…");
    myWindow.getContentPanel().add(myLabel);

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

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

`WindowDestroyer` is a programmer-defined class.

Allows the program to respond to the event of a user clicking in the close box.

The WindowDestroyer Class

```
public class WindowDestroyer extends WindowAdapter
{
  public void windowClosing(WindowEvent e)
  {
    System.exit(0);
  }
}
```

When a window closing event occurs, this method will be called and the program will quit.

`WindowAdapter` is a class that includes all the methods required for window events.

The Results of the Simple Demo Program

```
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…");
    myWindow.getContentPanel().add(myLabel);

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

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

The window will not show up on the screen without a line like this one.

Window Listeners

- Must have seven methods
- Each method is associated with a window event, such as a window closing
- Can inherit all seven methods from `WindowAdapter` and override some methods
- `WindowDestroyer` (in the Simple Demo Program) inherits from `WindowAdapter` and overrides only the `windowClosing` method.

A Better Version of the Simple Demo Program

- Separate class for the window
- `FirstWindow` class inherits from `JFrame`
- `main` method creates two instances of `FirstWindow`
- Each window has its own listener
- `setVisible` called from `main` for each window object

The `FirstWindow` Class

```
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…");
    getContentPanel().add(myLabel);

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

Inherits from `JFrame`

Calls constructor of base class

`setSize`, `getContentPanel`, and `addWindowListener` methods are inherited from `JFrame`

Methods of the `JFrame` Class

- `JFrame(String title)`

  constructor for creating a JFrame with a title
- `Container getContentPane()`

  returns the content pane of the JFrame, which has the add method for adding components
- `void setBackgroundColor(Color c)`
- `void setForegroundColor(Color c)`
- `void setSize(int width, int height)`
- `void setVisible(boolean b)`
- `void show()`

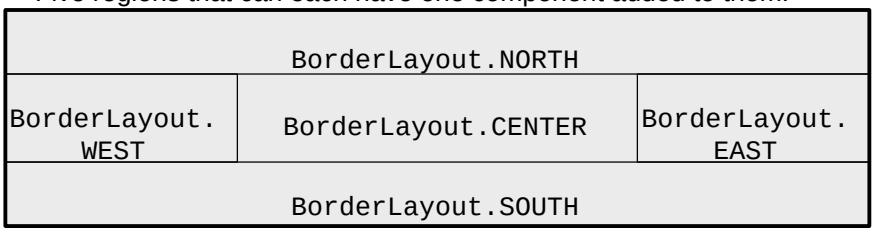
  sets visible and brings to front

Layout Managers

- Layout Manager—an object that decides how components will be arranged in a container
- Used because containers can change size
- Some types of layout managers:
  - » `BorderLayout`
  - » `FlowLayout`
  - » `GridLayout`
- Each type of layout manager has rules about how to rearrange components when the size or shape of the container changes.

The Border Layout Manager

Five regions that can each have one component added to them:




```
content.setLayout(new BorderLayout());
. . .
content.add(label1, BorderLayout.NORTH);
```

The CENTER region grows the most when the container grows and shrinks the most when the container shrinks

The Flow Layout Manager

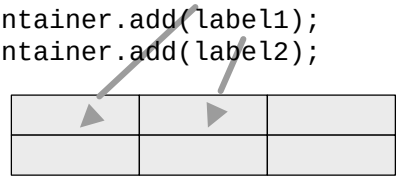
- The simplest layout manager
- Displays components from left to right in the order they are added to the container
- Add method has one parameter which is the component to add

```
Container content = getContentPane();
content.setLayout(new FlowLayout());
JLabel label1 = new JLabel("First label here");
content.add(label1);
JLabel label2 = new JLabel("Second label there");
content.add(label2);
```


The Grid Layout Manager

- Specify a number of rows and columns
- All regions in the grid are equal size
- When the container changes size, each region grows or shrinks by the same amount

```
aContainer.setLayout(new GridLayout(2, 3));
. . .
aContainer.add(label1);
aContainer.add(label2);
```



Creates a grid layout with two rows and three columns.

Rows are filled before columns.

Buttons and ActionListeners

Basic steps for using a button in a Swing program:

- Create a Button object
- Add the Button object to a container
- Create an `ActionListener` object that has an `actionPerformed` method
- Register the listener for the Button object

The following slides show an example of each step.

Create a Button Object and Add the Button to a Container

```
JButton stopButton = new JButton("Red");
```

`JButton` is a predefined Swing class for buttons.

String that will appear on the button

```
contentPane.add(stopButton);
```

The button will be added to this container.

This example uses the Flow Layout so the add method needs only one parameter.

Create an **`ActionListener`** Object

Make a class into an ActionListener:

- Add the phrase **`implements ActionListener`** to the beginning of the class definition:

```
public class ButtonDemo extends JFrame
    implements ActionListener
{   . . .
```

- Define a method named **`actionPerformed`**

```
public void actionPerformed(ActionEvent e)
{   . . .
```


The **`actionPerformed`** Method

- An `actionPerformed` method must have only **one** parameter
- The parameter **must** be of type `ActionEvent`

The parameter can be used to find the command for the `ActionEvent`:

```
public void actionPerformed(ActionEvent e)
{
  if (e.getActionCommand().equals("Red"))
    . . .
}
```

Register the Listener for the Button Object

- If a button has no listener registered for it, there will be no response when the user clicks on the button.
- An example of registering a listener for a button:

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

`this` refers to the object that includes this code in a method. In this example the object is a `JFrame` class that implements `ActionListener`.

Container Classes

A container class can have components added to it.

Every Swing container class has an `add` method.

Some commonly used container classes are:

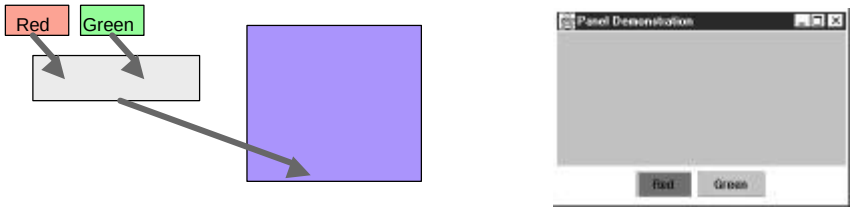
- `JPanel`
- `Container`
- Content pane of a `JFrame`

The following slides have information about each of these types of containers.

JPanel

- Used for hierarchical organization of GUIs:
  - » A panel can contain other components
  - » A panel can be added to another container

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout());
buttonPanel.add(stopButton);
buttonPanel.add(goButton);
contentPane.add(buttonPanel, BorderLayout.SOUTH);
```



The `Container` Class

- Any descendant of the `Container` class can have components added to it.
- Need to import the AWT library when using Container because it is not part of the Swing library:

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

- `JComponent` is derived from the `Container` class
- Components in the container are arranged by the container's layout manager

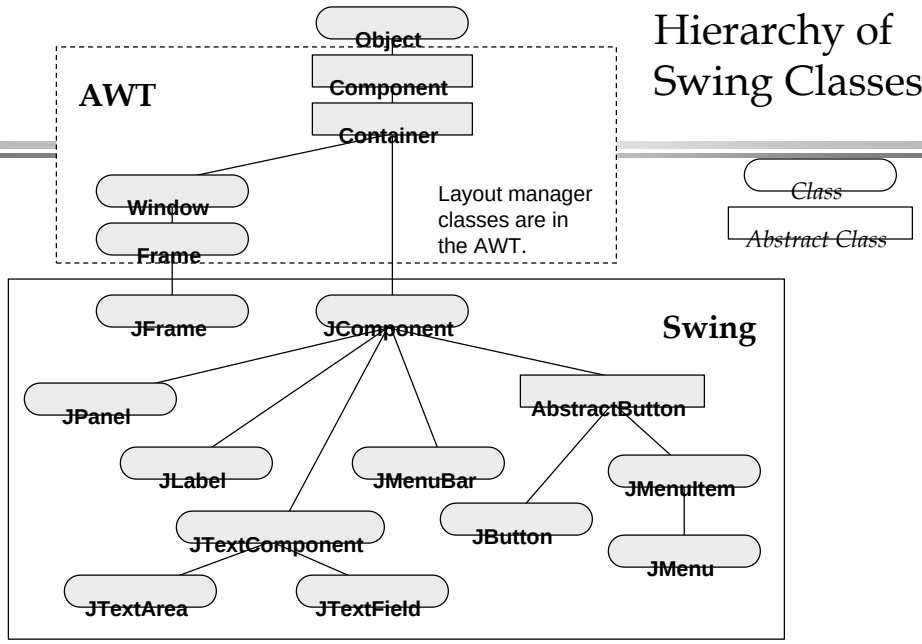
Content Pane of a `JFrame`

- Components are added to the content pane of a `JFrame` rather than directly to the `JFrame`
- The method `getContentPane` returns a reference to the content pane, which is treated as type `Container`

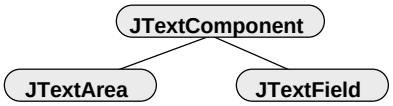
```
Container contentPane = getContentPane();
JLabel label = new JLabel("blue");
contentPane.add(label);
```

- For containers other than `JFrame` used in this book, `getContentPane` is not used

Hierarchy of Swing Classes



Text I/O for GUIs



Text fields and text areas

- » **`getText`** method retrieves text in component
- » **`setText`** changes text in component

If `memo1` is a `String` and `theText` is either a `JTextField` or a `JTextArea`, then you could write:

```
memo1 = theText.getText();
theText.setText("Hi Mom");
```


`JTextField` and **`JTextArea`**

- Both inherit from `JTextComponent`
- Both have `setText` and `getText` methods
- Both can have initializing text as parameter to constructor

- `JTextField` can only have one line of text
- `JTextArea` can have many lines of text
- `JTextArea` can have scroll bars

Big enough to hold 40 **m** characters

```
JTextField someText = new JTextField(40);
JTextArea someMoreText = new JTextArea(10, 40);
```

Big enough to hold 10 lines where each line can hold 40 **m** characters

Inputting and Outputting Numbers

To get an `int` from a `TextArea` or `TextField`:

- Get a string using `getText`
- Trim extra white space using `trim`
- Convert the String to an int using parseInt

```
int n = Integer.parseInt(field.getText().trim());
```


Inputting and Outputting Numbers

To get an `int` from a `TextArea` or `TextField`:

- Get a `String` using `getText`
- Trim extra white space using `trim`
- Convert the `String` to an `int` using `parseInt`

```
int n = Integer.parseInt(field.getText().trim());
```

To put an `int` into a `TextArea` or `TextField`:

- Convert the `int` to a `String` using `toString`
- Put the `String` in the text component using `setText`

```
field.setText(Integer.toString(total));
```

Inner Classes

- Classes within classes
- Advantages of inner classes:
  - » Make the outer class more self-contained
  - » Easily accessed by outer class
  - » Name is local to outer class so it can have the same name as another class
- Frequently used as listeners

Summary

- GUIs (Graphical User Interfaces) are programmed using event-driven programming.
- The class `JFrame` is used to create a windowing GUI.
- A button is an object of class `JButton`.
- The `add` method of a container can be used to add components to the container.
- Comonents are added to the content pane of a JFrame rather than directly to the JFrame.
- A panel is a container object that is used to group components inside of a larger container.
- Text fields and text areas are used for text input and output in a GUI constructed with Swing.