Chapter 17

Swing I

Introduction to Swing

- A GUI (graphical user interface) is a windowing system that interacts with the user
- The Java AWT (Abstract Window Toolkit) package is the original Java package for creating GUIs
- The Swing package is an improved version of the AWT
  - However, it does not completely replace the AWT
  - Some AWT classes are replaced by Swing classes, but other AWT classes are needed when using Swing
- Swing GUIs are designed using a form of object-oriented programming known as event-driven programming

Events

- Event-driven programming is a programming style that uses a signal-and-response approach to programming
- An event is an object that acts as a signal to another object known as a listener
- The sending of an event is called firing the event
  - The object that fires the event is often a GUI component, such as a button that has been clicked

Listeners

- A listener object performs some action in response to the event
  - A given component may have any number of listeners
  - Each listener may respond to a different kind of event, or multiple listeners might respond to the same events
Exception Objects

- An exception object is an event
  - The throwing of an exception is an example of firing an event
- The listener for an exception object is the `catch` block that catches the event

Event Handlers

- A listener object has methods that specify what will happen when events of various kinds are received by it
  - These methods are called *event handlers*
- The programmer using the listener object will define or redefine these event-handler methods

Event Firing and an Event Listener

- Event-driven programming is very different from most programming seen up until now
  - So far, programs have consisted of a list of statements executed in order
  - When that order changed, whether or not to perform certain actions (such as repeat statements in a loop, branch to another statement, or invoke a method) was controlled by the logic of the program
Event-Driven Programming

- In event-driven programming, objects are created that can fire events, and listener objects are created that can react to the events.
- The program itself no longer determines the order in which things can happen.
  - Instead, the events determine the order.

A Simple Window

- A simple window can consist of an object of the JFrame class.
  - A JFrame object includes a border and the usual three buttons for minimizing, changing the size of, and closing the window.
  - The JFrame class is found in the javax.swing package.
  - JFrame firstWindow = new JFrame();
- A JFrame can have components added to it, such as buttons, menus, and text labels.
  - These components can be programmed for action.
    - JFrame.add(endButton);
  - It can be made visible using the setVisible method.
    - firstWindow.setVisible(true);

A First Swing Demonstration (Part 1 of 4)

- A JFrame that can be made visible using the setVisible method.
  - It can be made visible using the setVisible method.
    - JFrame firstWindow = new JFrame();
  - It can be made visible using the setVisible method.
    - firstWindow.setVisible(true);
Some Methods in the Class JFrame (Part 1 of 3)

- The class JFrame is in the javax.swing package.
- public JFrame()
  Constructor that creates an object of the class JFrame.
- public JFrame(String title)
  Constructor that creates an object of the class JFrame with the title given as the argument.

(continued)
Some Methods in the Class JFrame (Part 2 of 3)

```java
public void setDefaultCloseOperation(int operation)
```
Sets the action that will happen by default when the user clicks the close-window button. The argument should be one of the following defined constants:
- JFrame.DO NOTHING ON CLOSE: Do nothing. The JFrame does nothing, but if there are any registered window listeners, they are invoked. (Window listeners are explained in Chapter 3.)
- JFrame.EXIT ON CLOSE: Exit the application using the System.exit method. (Do not use this for frames in accidents.)
- JFrame.HIDE ON CLOSE: Hide and dispose of the frame after invoking any registered window listeners. When a window is disposed it is eliminated but the program does not end. To end the program, you use the next constant as an argument to setDefaultCloseOperation.
- JFrame.EXIT ON CLOSE: Exit the application using the System.exit method. (Do not use this for frames in accidents.)
- JFrame.EXIT ON CLOSE: Exit the application using the System.exit method. (Do not use this for frames in accidents.)

Throws an IllegalArgumentException if the argument is not one of the values listed above.

```java
public void setSize(int width, int height)
```
Sets the size of the calling frame so that it has the width and height specified. Pixels are the units of length used.

(continued)

Pixels and the Relationship between Resolution and Size

- A `pixel` is the smallest unit of space on a screen
  - Both the size and position of Swing objects are measured in pixels
  - The more pixels on a screen, the greater the screen resolution
- A high-resolution screen of fixed size has many pixels
  - Therefore, each one is very small
- A low-resolution screen of fixed size has fewer pixels
  - Therefore, each one is much larger
- Therefore, a two-pixel figure on a low-resolution screen will look larger than a two-pixel figure on a high-resolution screen

Pitfall: Forgetting to Program the Close-Window Button

- The following lines from the `FirstSwingDemo` program ensure that when the user clicks the close-window button, nothing happens
  ```java
  firstWindow.setDefaultCloseOperation(JFrame.DO NOTHING ON CLOSE);
  ```
- If this were not set, the default action would be `JFrame.HIDE ON CLOSE`
  - This would make the window invisible and inaccessible, but would not end the program
  - Therefore, given this scenario, there would be no way to click the "Click to end program" button
- Note that the close-window and other two accompanying buttons are part of the `JFrame` object, and not separate buttons
Buttons

- A button object is created from the class `JButton` and can be added to a `JFrame`
  - The argument to the `JButton` constructor is the string that appears on the button when it is displayed
    ```java
    JButton endButton = new JButton("Click to end program.");
    firstWindow.add(endButton);
    ```

Action Listeners and Action Events

- Clicking a button fires an event
  - The event object is "sent" to another object called a listener
    - This means that a method in the listener object is invoked automatically
    - Furthermore, it is invoked with the event object as its argument
  - In order to set up this relationship, a GUI program must do two things
    1. It must specify, for each button, what objects are its listeners, i.e., it must register the listeners
    2. It must define the methods that will be invoked automatically when the event is sent to the listener

```java
EndingListener buttonEar = new EndingListener();
endButton.addActionListener(buttonEar);
```

- Above, a listener object named `buttonEar` is created and registered as a listener for the button named `endButton`
  - Note that a button fires events known as action events, which are handled by listeners known as action listeners
### Action Listeners and Action Events

```java
public void actionPerformed(ActionEvent e) {
    System.exit(0);
}
```

- The `EndingListener` class defines its `actionPerformed` method as above
  - When the user clicks the `endButton`, an action event is sent to the action listener for that button
  - The `EndingListener` object `buttonEars` is the action listener for `endButton`
  - The action listener `buttonEars` receives the action event as the parameter `e` to its `actionPerformed` method, which is automatically invoked
  - Note that `e` must be received, even if it is not used

### Pitfall: Changing the Heading for `actionPerformed`

- When the `actionPerformed` method is implemented in an action listener, its header must be the one specified in the `ActionListener` interface
  - It is already determined, and may not be changed
  - Not even a throws clause may be added

```java
public void actionPerformed(ActionEvent e) {
    // Method implementation
}
```

- The only thing that can be changed is the name of the parameter, since it is just a placeholder
  - Whether it is called `e` or something else does not matter, as long as it is used consistently within the body of the method

### Tip: Ending a Swing Program

- GUI programs are often based on a kind of infinite loop
  - The windowing system normally stays on the screen until the user indicates that it should go away
- If the user never asks the windowing system to go away, it will never go away
- In order to end a GUI program, `System.exit` must be used when the user asks to end the program
  - It must be explicitly invoked, or included in some library code that is executed
  - Otherwise, a Swing program will not end after it has executed all the code in the program

### A Better Version of Our First Swing GUI

- A better version of `FirstWindow` makes it a derived class of the class `JFrame`
  - This is the normal way to define a windowing interface
- The constructor in the new `FirstWindow` class starts by calling the constructor for the parent class using `super();`
  - This ensures that any initialization that is normally done for all objects of type `JFrame` will be done
- Almost all initialization for the window `FirstWindow` is placed in the constructor for the class
- Note that this time, an anonymous object is used as the action listener for the `endButton`
The Normal Way to Define a `JFrame` (Part 1 of 4)

```java
import javax.swing.JFrame;
import javax.swing.JButton;

public class FirstWindow extends JFrame {
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public FirstWindow() {
        super();
        setSize(WIDTH, HEIGHT);
        setTitle("First Window Class");
    }
}

(continued)
```

The Normal Way to Define a `JFrame` (Part 2 of 4)

```java
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

JButton endButton = new JButton("Click to end program.");
endButton.addActionListener(new EndListener());
add(endButton);

This is the file FirstWindow.java.

(continued)
```

The Normal Way to Define a `JFrame` (Part 3 of 4)

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

(continued)
```

The Normal Way to Define a `JFrame` (Part 4 of 4)

RESULTING GUI

```
Click to end program.
```

(continued)
Labels

- A label is an object of the class JLabel
  - Text can be added to a JFrame using a label
  - The text for the label is given as an argument when the JLabel is created
  - The label can then be added to a JFrame

```java
JLabel greeting = new JLabel("Hello");
add(greeting);
```

Color

- In Java, a color is an object of the class Color
  - The class Color is found in the java.awt package
  - There are constants in the Color class that represent a number of basic colors
- A JFrame can not be colored directly
  - Instead, a program must color something called the content pane of the JFrame
  - Since the content pane is the "inside" of a JFrame, coloring the content pane has the effect of coloring the inside of the JFrame
  - Therefore, the background color of a JFrame can be set using the following code:

```java
getContentPane().setBackground(Color.);
```

The Color Constants

<table>
<thead>
<tr>
<th>Color</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>MAGENTA</td>
</tr>
<tr>
<td>BLUE</td>
<td>ORANGE</td>
</tr>
<tr>
<td>CYAN</td>
<td>PINK</td>
</tr>
<tr>
<td>DARK_GRAY</td>
<td>RED</td>
</tr>
<tr>
<td>GRAY</td>
<td>WHITE</td>
</tr>
<tr>
<td>GREEN</td>
<td>YELLOW</td>
</tr>
</tbody>
</table>

The class Color is in the java.awt package.

A JFrame with Color (Part 1 of 4)

```java
import javax.swing.JFrame;
import java.awt.Color;

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

    public ColoredWindow(Color theColor)
    {
        super("No Change for Color");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```
Containers and Layout Managers

• Multiple components can be added to the content pane of a JFrame using the add method
  – However, the add method does not specify how these components are to be arranged
• To describe how multiple components are to be arranged, a layout manager is used
  – There are a number of layout manager classes such as BorderLayout, FlowLayout, and GridLayout
  – If a layout manager is not specified, a default layout manager is used
Border Layout Managers

- A **BorderLayout** manager places the components that are added to a **JFrame** object into five regions
  - These regions are: **BorderLayout.NORTH**, **BorderLayout.SOUTH**, **BorderLayout.EAST**, **BorderLayout.WEST**, and **BorderLayout.CENTER**
- A **BorderLayout** manager is added to a **JFrame** using the **setLayout** method
  - For example:
    ```java
    setDefaultCloseOperation(new BorderLayout());
    ```

The **BorderLayout** Manager (Part 1 of 4)

```java
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.BorderLayout;

public class BorderLayoutFrame extends JFrame {
    public static final int WIDTH = 500;
    public static final int HEIGHT = 400;

    public BorderLayoutFrame() {
        super("BorderLayout Demonstration");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```
(continued)

The **BorderLayout** Manager (Part 2 of 4)

```java
setLayout(new BorderLayout());
JLabel label1 = new JLabel("First label");
add(label1, BorderLayout.NORTH);
JLabel label2 = new JLabel("Second label");
add(label2, BorderLayout.SOUTH);
JLabel label3 = new JLabel("Third label");
add(label3, BorderLayout.CENTER);
```
(continued)

The **BorderLayout** Manager (Part 3 of 4)

```java
This is the file BorderLayoutDemo.java.
public class BorderLayoutDemo {
    public static void main(String[] args) {
        BorderLayoutFrame gui = new BorderLayoutFrame();
        gui.setVisible(true);
    }
}
```
(continued)
**Border Layout Managers**

- The previous diagram shows the arrangement of the five border layout regions.
  - Note: None of the lines in the diagram are normally visible.
- When using a `BorderLayout` manager, the location of the component being added is given as a second argument to the `add` method.
  - `add(label1, BorderLayout.NORTH)`;
  - Components can be added in any order since their location is specified.

**Flow Layout Managers**

- The `FlowLayout` manager is the simplest layout manager.
  - `setLayout(new FlowLayout());`
  - It arranges components one after the other, going from left to right.
  - Components are arranged in the order in which they are added.
- Since a location is not specified, the `add` method has only one argument when using the `FlowLayoutManager`.
  - `add(label1);`
Grid Layout Managers

- A `GridLayout` manager arranges components in a two-dimensional grid with some number of rows and columns:
  
  ```java
  setLayout(new GridLayout(rows, columns));
  ```
  
  - Each entry is the same size
  - The two numbers given as arguments specify the number of rows and columns
  - Each component is stretched so that it completely fills its grid position

  Note: None of the lines in the diagram are normally visible

Grid Layout Managers

- When using the `GridLayout` class, the method `add` has only one argument:
  
  ```java
  add(label1);
  ```
  
  - Items are placed in the grid from left to right
  - The top row is filled first, then the second, and so forth
  - Grid positions may not be skipped

- Note the use of a `main` method in the GUI class itself in the following example:
  
  - This is often a convenient way of demonstrating a class

The `GridLayout` Manager (Part 1 of 4)

```java
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.GridLayout;

public class GridLayoutFrame extends JFrame {
    public static final int WIDTH = 500;
    public static final int HEIGHT = 400;

    public static void main(String[] args) {
        GridLayoutFrame gui = new GridLayoutFrame(2, 3);
        gui.setVisible(true);
    }
}
```

The `GridLayout` Manager (Part 2 of 4)

```java
public GridLayoutFrame(int rows, int columns)
{
    super();
    setSize(WIDTH, HEIGHT);
    setTitle("GridLayout Demonstration");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new GridLayout(rows, columns));
    JLabel label1 = new JLabel("First label");
    add(label1);
}
```
Some Layout Managers

<table>
<thead>
<tr>
<th>LAYOUT MANAGER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlowLayout</td>
<td>Displays components from left to right in the order in which they are added to the container.</td>
</tr>
<tr>
<td>BorderLayout</td>
<td>Displays the components in five areas: north, south, east, west, and center. You specify the area a component goes into in a second argument of the add method.</td>
</tr>
<tr>
<td>GridLayout</td>
<td>Lays out components in a grid, with each component stretched to fill its box in the grid.</td>
</tr>
</tbody>
</table>

Panels

- A GUI is often organized in a hierarchical fashion, with containers called panels inside other containers.
- A panel is an object of the JPanel class that serves as a simple container:
  - It is used to group smaller objects into a larger component (the panel).
  - One of the main functions of a JPanel object is to subdivide a JFrame or other container.
Panels

• Both a `JFrame` and each panel in a `JFrame` can use different layout managers
  – Additional panels can be added to each panel, and each panel can have its own layout manager
  – This enables almost any kind of overall layout to be used in a GUI

```java
setLayout(new BorderLayout());
JPanel somePanel = new JPanel();
somePanel.setLayout(new FlowLayout());
```

• Note in the following example that panel and button objects are given color using the `setBackground` method without invoking `getContentPane`
  – The `getContentPane` method is only used when adding color to a `JFrame`

```java
display 17.11 Using panels
1 import javax.swing.JFrame;
2 import javax.swing.JPanel;
3 import javax.awt.BorderLayout;
4 import javax.awt.GridBagLayout;
5 import javax.awt.GridBagConstraints;
6 import javax.awt.Color;
7 import javax.swing.JButton;
8 import javax.awt.event.ActionListener;
9 import javax.awt.event.ActionEvent;
10
11 public class PanelDemo extends JFrame implements ActionListener
12 {
13   public static final int WIDTH = 300;
14   public static final int HEIGHT = 200;
15
16   private JPanel redPanel;
17   private JPanel whitePanel;
18   private JPanel bluePanel;
19
20   public static void main(String[] args)
21   {
22     JFrame gui = new PanelDemo();
23     gui.setVisible(true);
24
25     super("Panel Demonstration");
26     setSize(WIDTH, HEIGHT);
27     setLocationRelativeTo(JFrame.EXIT_ON_CLOSE);
28     setLayout(new BorderLayout());
```
Using Panels (Part 4 of 8)

```java
36    bluePanel = new JPanel();
37    bluePanel.setBackground(Color.LIGHT_GRAY);
38    biggerPanel.add(bluePanel);
39    add(biggerPanel, BorderLayout.CENTER);
40    JPanel buttonPanel = new JPanel();
41    buttonPanel.setBackground(Color.LIGHT_GRAY);
42    buttonPanel.setLayout(new FlowLayout());
43    JButton redButton = new JButton("Red");
44    redButton.setBackground(Color.RED);
45    redButton.addActionListener(this);
46    buttonPanel.add(redButton);
```

An object of the class JPanelDemo is the action
listener for the buttons in this object.

(continued)

Using Panels (Part 5 of 8)

```java
47    JButton whiteButton = new JButton("White");
48    whiteButton.setBackground(Color.WHITE);
49    whiteButton.addActionListener(this);
50    buttonPanel.add(whiteButton);
51    JButton blueButton = new JButton("Blue");
52    blueButton.setBackgraund(Color.BLUE);
53    blueButton.addActionListener(this);
54    buttonPanel.add(blueButton);
55    add(buttonPanel, BorderLayout.SOUTH);
```

(continued)

Using Panels (Part 6 of 8)

```java
57    public void actionPerformed(ActionEvent e)
58    {
59        String buttonString = e.getActionCommand();
60        if (buttonString.equals("Red"))
61            redPanel.setBackground(Color.RED);
62        else if (buttonString.equals("White"))
63            whitePanel.setBackground(Color.WHITE);
64        else if (buttonString.equals("Blue"))
65            bluePanel.setBackground(Color.BLUE);
66        else
67            System.out.println("Unexpected error.");
68    }
```

(continued)

Using Panels (Part 7 of 8)

```java
```

(continued)
Using Panels (Part 8 of 8)

**The Container Class**

- Any class that is a descendant class of the class `Container` is considered to be a container class
  - The `Container` class is found in the `java.awt` package, not in the Swing library
- Any object that belongs to a class derived from the `Container` class (or its descendents) can have components added to it
- The classes `JFrame` and `JPanel` are descendant classes of the class `Container`
  - Therefore they and any of their descendents can serve as a container

**The JComponent Class**

- Any descendant class of the class `JComponent` is called a `component class`
  - Any `JComponent` object or `component` can be added to any container class object
  - Because it is derived from the class `Container`, a `JComponent` can also be added to another `JComponent`

**Objects in a Typical GUI**

- Almost every GUI built using Swing container classes will be made up of three kinds of objects:
  1. The container itself, probably a panel or window-like object
  2. The components added to the container such as labels, buttons, and panels
  3. A layout manager to position the components inside the container
Tip: Code a GUI's Look and Actions Separately

- The task of designing a Swing GUI can be divided into two main subtasks:
  1. Designing and coding the appearance of the GUI on the screen
  2. Designing and coding the actions performed in response to user actions

- In particular, it is useful to implement the `actionPerformed` method as a stub, until the GUI looks the way it should.

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

- This philosophy is at the heart of the technique used by the Model-View-Controller pattern

The Model-View-Controller Pattern

- A menu is an object of the class `JMenu`
- A choice on a menu is called a menu item, and is an object of the class `JMenuItem`
  - A menu can contain any number of menu items
  - A menu item is identified by the string that labels it, and is displayed in the order to which it was added to the menu
- The `add` method is used to add a menu item to a menu in the same way that a component is added to a container object
Menu Bars, Menus, and Menu Items

- The following creates a new menu, and then adds a menu item to it
  ```java
  JMenu diner = new JMenu("Daily Specials");
  JMenuItem lunch = new JMenuItem("Lunch Specials");
  lunch.addActionListener(this);
  diner.add(lunch);
  ```
  – Note that the this parameter has been registered as an action listener for the menu item

Nested Menus

- The class JMenu is a descendent of the JMenuItem class
  - Every JMenu can be a menu item in another menu
  - Therefore, menus can be nested
- Menus can be added to other menus in the same way as menu items

Menu Bars and JFrame

- A menu bar is a container for menus, typically placed near the top of a windowing interface
- The add method is used to add a menu to a menu bar in the same way that menu items are added to a menu
  ```java
  JMenuBar bar = new JMenuBar();
  bar.add(diner);
  ```
- The menu bar can be added to a JFrame in two different ways
  1. Using the setJMenuBar method
     ```java
     setJMenuBar(bar);
     ```
  2. Using the add method – which can be used to add a menu bar to a JFrame or any other container

A GUI with a Menu (Part 1 of 8)
A GUI with a Menu (Part 2 of 8)

```java
public class MenuDemo extends JFrame implements ActionListener {

    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

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

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

A GUI with a Menu (Part 3 of 8)

```java
public class MenuDemo {

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

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

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

A GUI with a Menu (Part 4 of 8)

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

JMenu colorMenu = new JMenu("Add Colors");
JMenuItem redChoice = new JMenuItem("Red");
redChoice.addActionListener(this);
colorMenu.add(redChoice);
JMenuItem whiteChoice = new JMenuItem("White");
whiteChoice.addActionListener(this);
colorMenu.add(whiteChoice);
```

A GUI with a Menu (Part 5 of 8)

```java
JMenu item blueChoice = new JMenuItem("Blue");
blueChoice.addActionListener(this);
colorMenu.add(blueChoice);
JMenuBar bar = new JMenuBar();
bar.add(colorMenu);
setMenuBar(bar);
```
A GUI with a Menu (Part 6 of 8)

```java
public void actionPerformed(ActionEvent e)
{
    String buttonString = e.getActionCommand();
    if (buttonString.equals("Red"))
        redPanel.setBackground(Color.RED);
    else if (buttonString.equals("White"))
        whitePanel.setBackground(Color.WHITE);
    else if (buttonString.equals("Blue"))
        bluePanel.setBackground(Color.BLUE);
    else
        System.out.println("Unexpected error.");
}
(continued)
```

A GUI with a Menu (Part 7 of 8)

The `AbstractButton` and `Dimension` Classes

- The classes `JButton` and `JMenuItem` are derived classes of the abstract class named `AbstractButton`
  - All of their basic properties and methods are inherited from the class `AbstractButton`
- Objects of the `Dimension` class are used with buttons, menu items, and other objects to specify a size
  - The `Dimension` class is in the package `java.awt`
  - Note: `width` and `height` parameters are in pixels
The `setActionCommand` Method

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

```java
e.getActionCommand();
```

The `setActionCommand` Method

- The `setActionCommand` method can be used to change the action command for a component
  - This is especially useful when two or more buttons or menu items have the same default action command strings
    ```java
    JButton nextButton = new JButton("Next");
    nextButton.setActionCommand("Next Button");
    
    JMenuItem choose = new JMenuItem("Next");
    choose.setActionCommand("Next Menu Item");
    ```

Some Methods in the Class `AbstractButton` (Part 1 of 3)

- `public void setBackground(Color theColor)`
  - Sets the background color of this component.
- `public void addActionListener(ActionListener listener)`
  - Adds an `ActionListener`.
- `public void removeActionListener(ActionListener listener)`
  - Removes an `ActionListener`.
- `public void setActionCommand(String actionCommand)`
  - Sets the action command.

(continued)
Listeners as Inner Classes

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

Listeners as Inner Classes (Part 1 of 6)

```java
public class RedListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        redPanel.setBackground(Color.RED);
    }
}
```

Listeners as Inner Classes (Part 2 of 6)

```java
public class WhiteListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        whitePanel.setBackground(Color.WHITE);
    }
}
```
private class BlueListener implements ActionListener

public void actionPerformed(ActionEvent e)
{
    bluePanel.setBackground(Color.BLUE);
}

} //end of BlueListener inner class

public static void main(String[] args)
{
    InnerListenersDemo gui = new InnerListenersDemo();
    gui.setVisible(true);
    // (continued)

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

    JMenu colorMenu = new JMenu("Add Colors");
    JMenuItem redChoice = new JMenuItem("Red");
    redChoice.addActionListener(new RedListener());
    colorMenu.add(redChoice);
    // (continued)

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

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

    JMenuBar bar = new JMenuBar();
    bar.add(colorMenu);
    // (continued)
Text Fields

- A text field is an object of the class `JTextField`
  - It is displayed as a field that allows the user to enter a single line of text
    ```java
    private JTextField name;
    ... 
    name = new JTextField(NUMBER_OF_CHAR);
    ```
  - In the text field above, at least `NUMBER_OF_CHAR` characters can be visible

There is also a constructor with one additional `String` parameter for displaying an initial `String` in the text field
```java
JTextField name = new JTextField("Enter name here.", 30);
```

A Swing GUI can read the text in a text field using the `getText` method
```java
String inputString = name.getText();
```

The method `setText` can be used to display a new text string in a text field
```java
name.setText("This is some output");
```
A Text Field (Part 3 of 7)

```java
24    public TextFieldDemo()
25    {
26        super("Text Field Demo");
27        setSize(WIDTH, HEIGHT);
28        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29        setLocation(new GridLayout(2, 3));
30        JPanel namePanel = new JPanel();
31        namePanel.setLayout(new BorderLayout());
32        namePanel.setBackground(Color.WHITE);
33        name = new JTextField(NUMBER_OF_CHARS);
```

(continued)

A Text Field (Part 4 of 7)

```java
34        namePanel.add(name, BorderLayout.SOUTH);
35        JLabel nameLabel = new JLabel("Enter your name here:");
36        namePanel.add(nameLabel, BorderLayout.CENTER);
37        add(namePanel);
```

```java
38        JPanel buttonPanel = new JPanel();
39        buttonPanel.setLayout(new FlowLayout());
40        buttonPanel.setBackground(Color.PINK);
41        JButton actionButton = new JButton("Click me");
42        actionButton.addActionListener(this);
43        buttonPanel.add(actionButton);
44        JButton clearButton = new JButton("Clear");
45        clearButton.addActionListener(this);
46        buttonPanel.add(clearButton);
```

(continued)

A Text Field (Part 5 of 7)

```java
47        add(buttonPanel);
48    }
```

```java
49    public void actionPerformed(ActionEvent e)
50    {
51        String actionCommand = e.getActionCommand();
52        if (actionCommand.equals("Click me"))
53            name.setText("Hello : " + name.getText());
54        else if (actionCommand.equals("Clear"))
55            name.setText("");
56        else
57            name.setText("Unexpected error.");
58    }
```

(continued)

A Text Field (Part 6 of 7)

RESULTING GUI (when program is started and a name entered)

[Image of GUI showing "TextField Demo" with "Enter your name here:"
and "Click me" buttons, with a text field labeled with "Hello: Name"]

(continued)
A Text Field (Part 7 of 7)

Text Areas

- A text area is an object of the class JTextArea
  - It is the same as a text field, except that it allows multiple lines
  - Two parameters to the JTextArea constructor specify the minimum number of lines, and the minimum number of characters per line that are guaranteed to be visible
    ```java
    JTextArea theText = new JTextArea(5,20);
    ```
  - Another constructor has one additional String parameter for the string initially displayed in the text area
    ```java
    JTextArea theText = new JTextArea("Enter\ntext here.", 5, 20);
    ```

Text Areas

- The line-wrapping policy for a JTextArea can be set using the method setLineWrap
  - The method takes one boolean type argument
  - If the argument is true, then any additional characters at the end of a line will appear on the following line of the text area
  - If the argument is false, the extra characters will remain on the same line and not be visible
    ```java
    theText.setLineWrap(true);
    ```

Text Fields and Text Areas

- A JTextField or JTextArea can be set so that it can not be changed by the user
  ```java
  theText.setEditable(false);
  ```
  - This will set theText so that it can only be edited by the GUI program, not the user
  - To reverse this, use true instead (this is the default)
    ```java
    theText.setEditable(true);
    ```
Tip: Labeling a Text Field

• In order to label one or more text fields:
  – Use an object of the class JLabel
  – Place the text field(s) and label(s) in a JPanel
  – Treat the JPanel as a single component

Numbers of Characters Per Line

• The number of characters per line for a JTextField or JTextArea object is the number of em spaces
  • An em space is the space needed to hold one uppercase letter M
    – The letter M is the widest letter in the alphabet
    – A line specified to hold 20 M’s will almost always be able to hold more than 20 characters

Tip: Inputting and Outputting Numbers

• When attempting to input numbers from any Swing GUI, input text must be converted to numbers
  – If the user enters the number 42 in a JTextField, the program receives the string "42" and must convert it to the integer 42
• The same thing is true when attempting to output a number
  – In order to output the number 42, it must first be converted to the string "42"

The Class JTextComponent

• Both JTextField and JTextArea are derived classes of the abstract class JTextComponent
• Most of their methods are inherited from JTextComponent and have the same meanings
  – Except for some minor redefinitions to account for having just one line or multiple lines
Some Methods in the Class `JTextComponent` (Part 1 of 2)

`public String getText()`
Returns the text that is displayed by this text component.

`public boolean isEditable()`
Returns true if the user can write in this text component. Returns false if the user is not allowed to write in this text component.

(continued)

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

`public void setBackground(Color theColor)`
Sets the background color of this text component.

`public void setEditable(boolean argument)`
If argument is true, then the user is allowed to write in this text component. If argument is false, then the user is not allowed to write in this text component.

`public void setText(String text)`
Sets the text that is displayed by this text component to be the specified text.

A Swing Calculator

• A GUI for a simple calculator keeps a running total of numbers
  – The user enters a number in the text field, and then clicks either + or –
  – The number in the text field is then added to or subtracted from the running total, and displayed in the text field
  – This value is kept in the instance variable `result`
  – When the GUI is first run, or when the user clicks the `Reset` button, the value of `result` is set to zero

A Swing Calculator

• If the user enters a number in an incorrect format, then one of the methods throws a `NumberFormatException`
  – The exception is caught in the catch block inside the `actionPerformed` method
  – Note that when this exception is thrown, the value of the instance variable `result` is not changed
Display 17.19  A Simple Calculator

```java
1 import javax.swing.JFrame;
2 import javax.swing.JTextField;
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
12 /*
13 A simplified calculator.
14 The only operations are addition and subtraction.
15 */
16 public class Calculator extends JFrame
17 {
18     public static final int WIDTH = 480;
19     public static final int HEIGHT = 286;
20     public static final int NUMBER_OF_DIGITS = 39;
21     private JTextField iofield;
22     private double result = 0.0;
23     public static void main(String[] args)
24     {
25         Calculator calculator = new Calculator();
26         calculator.setVisible(true);
27     }
28     public Calculator()
29     {
30         setTitle("Simplified Calculator");
31         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32         setSize(WIDTH, HEIGHT);
33         setLayout(new BorderLayout());
34         JPanel textPanel = new JPanel();
35         textPanel.setLayout(new FlowLayout());
36         iofield =
37             new JTextField("Enter numbers here.", NUMBER_OF_DIGITS);
38         iofield.setBackground(Color.WHITE);
39         textPanel.add(iofield);
40         add(textPanel, BorderLayout.NORTH);
41     }
42     Button addButton = new JButton("+");
43     Button subtractButton = new JButton("-");
44     Button resetButton = new JButton("Reset");
45     buttonPanel.add(addButton);
46     buttonPanel.add(subtractButton);
47     buttonPanel.add(resetButton);
48     add(subtractButton, ActionListener(this));
49     add(resetButton, ActionListener(this));
50     add(addButton, ActionListener(this));
51     setVisible(true);
52 }
53 ```

A Simple Calculator (Part 5 of 11)

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

A NumberFormatException does not need to be declared or caught in a catch block.

(continued)

A Simple Calculator (Part 6 of 11)

```java
else if (actionCommand.equals("-"))
{
    result = result - stringToDouble(textField.getText());
    textField.setText(Double.toString(result));
```

(continued)

A Simple Calculator (Part 7 of 11)

```java
else if (actionCommand.equals("Reset"))
{
    result = 0.0;
    textField.setText("0.0");
```

(continued)

A Simple Calculator (Part 8 of 11)

```java
private static double StringToDouble(String stringObject)
{
    return Double.parseDouble(stringObject.trim());
```

(continued)
Uncaught Exceptions

- In a Swing program, throwing an uncaught exception does not end the GUI
  - However, it may leave it in an unpredictable state
- It is always best to catch any exception that is thrown even if all the catch block does is output an error message, or ask the user to reenter some input