

Chapter 11

GUIs and Event-Driven Programming

Swing is a Java package that provides a set of classes for creating graphical user interfaces (GUIs). This chapter explains how to use the basic Swing components to create applications with a GUI. Components covered in this chapter include frames, panels, labels, buttons, combo boxes, and text fields.

TIP The applications created prior to this chapter had a text-based, or console, interface.

What is a GUI?

A *GUI* is a graphical user interface (sometimes pronounced “gooey”). An application written for the Microsoft Windows or Mac OSX Tiger operating system typically has a GUI. The GUI below includes a frame, also called a window, and a label:



Not only does a GUI use components such as frames, buttons, and text fields to communicate with the user, but GUIs are event-driven. An *event-driven application* executes code in response to events. An *event* is an interaction from the user, such as a button click. A GUI responds to an event by executing a method called an *event handler*. For example, when the user clicks a button, the application must be able to determine which button was clicked and then execute the code that relates to that button.

The Swing Package

The Swing API is part of the Java Foundation Classes (JFC) and contains numerous components for creating GUIs. This chapter focuses on the `javax.swing` package. Two classes in this package are `JFrame` and `JLabel`. The `JFrame` class is used to instantiate a frame. A *frame* is a window with a border, title, and buttons for minimizing, maximizing, and closing the frame. A frame is a top-level *container* for a GUI, which holds and displays all the other components of an interface in a *content frame*. One of the most common components in a content frame is a label. *Labels*, created with the `JLabel` class, are used to display text that cannot be changed by the user.

event-driven application

event handler

*javax.swing
frame*

container, content frame

labels

The swing package includes the JFrame class with the following methods:

Class JFrame (javax.swing.JFrame)

Method

`setDefaultLookAndFeelDecorated(boolean)`
sets the frames created after this class method is called to have Java Window decorations, such as borders and a title, when the `boolean` argument is `true`.

`setDefaultCloseOperation(class _ constant)`
sets the operation that occurs when the user clicks the Close button. The `class _ constant` argument is usually `JFrame.EXIT_ON_CLOSE`.

`getContentPane()`
returns a `Container` object corresponding to the content pane.

`setContentPane(Container contentPane)`
sets the content pane to `contentPane`. The `Container` class is the superclass for the `JPanel` class, a class for creating content panes.

`pack()`
sizes the frame so that all of its contents are at or above their preferred sizes.

`setVisible(boolean)`
displays the frame when the `boolean` argument is `true`.

JPanel content pane

A `JFrame` object uses a content pane to hold GUI components. A `JPanel` object is one choice for a simple content pane. The `JPanel` class includes the following methods for adding and removing components:

Class JPanel (javax.swing.JPanel)

Method

`add(Component GUIcomponent)`
adds a `GUIcomponent` to the content pane. When added, components are given an index value, with the first component at index 0.

`remove(int index)`
removes the component with index value `index`.

After adding components to the `JPanel` object, the content frame is added to the `JFrame` object using the `JFrame setContentPane()` method described above.

The swing package includes the `JLabel` class for creating labels that can be added to a content pane. `JLabels` have several constructors, two of which are described, in addition to the `setText()` method:

Class JLabel (javax.swing.JLabel)

Constructor/Method

JLabel(String str)

creates a JLabel object with text `str`.

JLabel(String str, align _ constant)

creates a JLabel object with text `str` and alignment `align _ constant` that can be set to JLabel class constants `LEADING` or `TRAILING`, which are left and right alignment, respectively.

setText(String str)

sets the text of the JLabel object to `str`.

The HelloWorldWithGUI1 application creates a frame and a content pane, adds the content pane to the frame, and then displays the frame:

TIP The HelloWorldWithGUI1 application output is shown on the first page of this chapter.

```
import javax.swing.*;

public class HelloWorldWithGUI1 {
    final static String LABEL_TEXT = "Hello, world!";
    JFrame frame;
    JPanel contentPane;
    JLabel label;

    public HelloWorldWithGUI1(){
        /* Create and set up the frame */
        frame = new JFrame("HelloWorldWithGUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane */
        contentPane = new JPanel();

        /* Create and add label */
        label = new JLabel(LABEL_TEXT);
        contentPane.add(label);

        /* Add content pane to frame */
        frame.setContentPane(contentPane);

        /* Size and then display the frame. */
        frame.pack();
        frame.setVisible(true);
    }

    /**
     * Create and show the GUI.
     */
    private static void runGUI() {
        JFrame.setDefaultLookAndFeelDecorated(true);

        HelloWorldWithGUI1 greeting = new HelloWorldWithGUI1();
    }

    public static void main(String[] args) {
        /* Methods that create and show a GUI should be
         run from an event-dispatching thread */
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                runGUI();
            }
        });
    }
}
```

The implementation of a Swing application is different from previous applications presented in this text. The controlling class contains a constructor and members in addition to the `main()` method. Therefore, the controlling class is actually used to instantiate an object, which implements the GUI.

thread The statement in the `main()` method runs the GUI from an event-dispatching thread. A *thread* is simply a process that runs sequentially from start to finish. GUIs should be invoked from an event-dispatching thread to ensure that each event-handler finishes executing before the next one executes. Thorough coverage of this topic is beyond the scope of this text. However, the code shown is needed in every application that implements a Swing GUI.

Review: Name – part 1 of 2

Create a Name application that displays your name in a label inside a JFrame.

The JButton Class

A button is a commonly used GUI component. A *button* can be clicked by the user to communicate with the application. For example, the `HelloWorldWithGUI2` application includes a button that when clicked either hides or displays text in the label:



The swing package includes the `JButton` class for creating buttons:

Class `JButton` (`javax.swing.JButton`)

Method

`JButton(String str)`

creates a `JButton` object displaying the text `str`.

`setActionCommand(String cmd)`

sets the name of the action performed when the button is clicked to `cmd`.

`getActionCommand()`

returns the name of the action that has been performed by the button.

`addActionListener(Object)`

adds an object that listens for the user to click this component.

Unlike a `JLabel`, a `JButton` can respond to interaction from the user.

Handling Events

listener

Swing components use listeners to determine if an event has occurred. A *listener* is an object that listens for action events. When an event is heard, the listener responds by executing an event handler named `actionPerformed()`. The `actionPerformed()` method has an `ActionEvent` parameter passed by the GUI when an event occurs. The `ActionEvent` object includes an action command. An *action command* is a string describing an event, or action.

action command

The `HelloWorldWithGUI2` application output is shown in the previous section. When the Hide button is clicked, the application GUI changes to display:



this

The `HelloWorldWithGUI2` application code is below. For this GUI, a `JButton` is created, the `JButton` action command is set, and the current object is the listener for `JButton` action events. Note the use of the keyword `this` for the listener object to indicate the `HelloWorldWithGUI2` object itself:

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

public class HelloWorldWithGUI2 implements ActionListener {
    final static String LABEL_TEXT = "Hello, world!";
    JFrame frame;
    JPanel contentPane;
    JLabel label;
    JButton button;

    public HelloWorldWithGUI2(){
        /* Create and set up the frame */
        frame = new JFrame("HelloWorldWithGUI2");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane */
        contentPane = new JPanel();

        /* Create and add label */
        label = new JLabel(LABEL_TEXT);
        contentPane.add(label);

        /* Create and add button */
        button = new JButton("Hide");
        button.setActionCommand("Hide");
        button.addActionListener(this);
        contentPane.add(button);

        /* Add content pane to frame */
        frame.setContentPane(contentPane);

        /* Size and then display the frame. */
        frame.pack();
        frame.setVisible(true);
    }
}
```

```

/**
 * Handle button click action event
 * pre: Action event is Hide or Show
 * post: Clicked button has different text, and label
 * displays message depending on when the button was clicked.
 */
public void actionPerformed(ActionEvent event) {
    String eventName = event.getActionCommand();

    if (eventName.equals("Hide")) {
        label.setText(" ");
        button.setText("Show");
        button.setActionCommand("Show");
    } else {
        label.setText(LABEL_TEXT);
        button.setText("Hide");
        button.setActionCommand("Hide");
    }
}

/**
 * Create and show the GUI.
 */
private static void runGUI() {
    JFrame.setDefaultLookAndFeelDecorated(true);

    HelloWorldWithGUI2 greeting = new HelloWorldWithGUI2();
}

public static void main(String[] args) {
    /* Methods that create and show a GUI should be
     run from an event-dispatching thread */
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            runGUI();
        }
    });
}
}

```

TIP awt stands for Abstract Windows Toolkit.

A class that uses a listener must implement an `ActionListener`. This is done by adding `implements ActionListener` to the class declaration and then defining an `actionPerformed()` method, which is the only method in the `ActionListener` interface. An `import java.awt.event` statement is required to import the package containing the `ActionListener` interface.

The GUI is implemented in the constructor. The segment of code that creates and adds the button does four things. First, a button that displays Hide is created. Second, because the user will see a Hide button the action command associated with this button should be "Hide". Third, a listener is needed to determine when the user clicks the button. The listener is set to the `HelloWorldWithGUI2` object itself (`this`). Fourth, the button is added to the content pane.

The `actionPerformed()` method is passed an `ActionEvent` argument by the GUI when the button is clicked. `ActionEvent` objects have a `getActionCommand()` method. This method returns the string assigned as the object's action command. For the Show/Hide button, the action command, as well as the button text, is changed each time the button is clicked.

Review: Name – part 2 of 2

Modify the Name application to display or hide your name depending on the button clicked by the user, similar to the HelloWorldWithGUI2 application.

Review: NumClicks

Create a NumClicks application that contains a button displaying how many times the user has clicked that button. The application interface should look similar to the following after the user has clicked the button 12 times:



For this application it is not necessary to set the action command. A call to the `getActionCommand()` method in the `actionPerformed()` method is also not needed.

Controlling Layout

layout

Layout refers to the arrangement of components. In a Swing GUI, the layout of a content pane can be controlled by adding borders, using a layout manager, and setting alignments.

borders

A border can be added to most components, including the content pane. An invisible, or empty, border can be used to add “padding” around a component to give it distance from other components. Adding an empty border to the content pane adds space between the components and the edges of the frame. For example, the content pane in the GUI on the left has no border, but the content pane on the right has an empty border of 20 pixels on the top, left, bottom, and right:

TIP *Pixel* stands for picture element and the number of pixels in a surface depends on the screen resolution.



An empty border in the content pane on the right provides padding between the components and the frame

layout manager

FlowLayout manager

A *layout manager* determines the order of components on a content pane. There are many layout managers to choose from, including `FlowLayout`, `BoxLayout`, and `GridLayout`. The *FlowLayout manager* places components one next to the other in a row. When a row becomes too long, a new row is started. The `FlowLayout` manager is the default manager. The HelloWorldWithGUI applications use this manager.

BoxLayout manager

TIP The BoxLayout manager also includes methods for adding “glue” and “rigid areas” to a layout to control placement of components.

The *BoxLayout manager* places components one after the other in a column, with one component per line. For example, the GUI below uses the BoxLayout manager:



The BoxLayout manager places components one after the other in a column

GridLayout manager

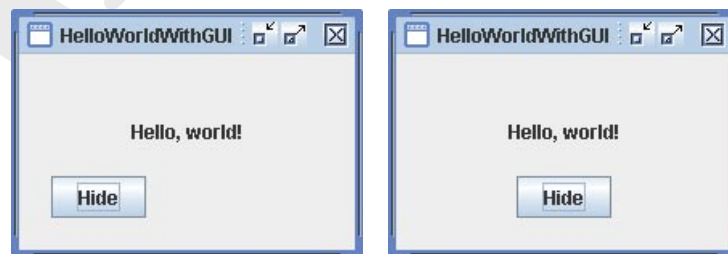
The *GridLayout manager* places components into a grid of rows and columns. The intersection of a row and column is called a *cell*. There is one component per cell in a GridLayout. The GUI below uses a GridLayout. An additional button was added to illustrate the grid:



The GridLayout manager places components into a grid of cells

alignment

Another factor that affects layout is alignment. *Alignment* refers to the placement of a component within a layout. For example, both GUIs below use a BoxLayout, have an empty border of 20, 20, 20, and 20 in the content pane, and a 20, 50, 20, 50 empty border around the label. The GUI on the left specifies no alignment for the components, and the GUI on the right center aligns the label and the button:



Alignment affects the placement of components within a layout

The HelloWorldWithGUI2 application modified to use a layout manager, borders, and alignment:

TIP Experiment with the numerous types of borders.

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

public class HelloWorldWithGUI2 implements ActionListener {
    final static String LABEL_TEXT = "Hello, world!";
    JFrame frame;
    JPanel contentPane;
    JLabel label;
    JButton button;

    public HelloWorldWithGUI2(){
        /* Create and set up the frame */
        frame = new JFrame("HelloWorldWithGUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a BorderLayout and
           empty borders */
        contentPane = new JPanel();
        contentPane.setLayout(new BorderLayout(contentPane,
                                              BorderLayout.PAGE_AXIS));
        contentPane.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

        /* Create and add label that is centered and
           has empty borders */
        label = new JLabel(LABEL_TEXT);
        label.setAlignmentX(JLabel.CENTER_ALIGNMENT);
        label.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));
        contentPane.add(label);

        /* Create and add button that is centered */
        button = new JButton("Hide");
        button.setAlignmentX(JButton.CENTER_ALIGNMENT);
        button.setActionCommand("Hide");
        button.addActionListener(this);
        contentPane.add(button);

        /* Add content pane to frame */
        frame.setContentPane(contentPane);

        /* Size and then display the frame. */
        frame.pack();
        frame.setVisible(true);
    }
}
```

...rest of HelloWorldWithGUI2 code

setLayout() method

The `JPanel` `setLayout()` method is used to specify a layout for the content pane. A `BoxLayout` object requires arguments for the content pane and the arrangement. To arrange components in a vertical line, the class constant `PAGE_AXIS` is specified.

setBorder() method

The `JPanel` `setBorder()` method is used to specify borders for the content pane. The `BorderFactory` class has many different kinds of borders to choose from. For invisible, or empty borders, the `createEmptyBorder()` method is used. The arguments specify the width of the top, left, bottom, and right of the border.

setAlignment() method

The `JLabel` and `JButton` `setAlignmentX()` methods are used to specify the vertical alignment of the components within the layout. Both classes include several alignment constants, including `CENTER_ALIGNMENT`.

The modified HelloWorldWithGUI2 application produces a GUI that looks similar to:



GridLayout manager

TIP The GridLayout manager places components in rows from left to right in the order that they are added to the content pane.

For GUIs with components that should be side by side in rows, the GridLayout manager may be a better choice. The GridLayout manager is specified in a statement similar to:

```
contentPane.setLayout(new GridLayout(0, 2, 10, 5))
```

The GridLayout object requires arguments for specifying the number of rows and columns and the space between columns and rows. If 0 is specified for either the rows or columns, the class creates an object with as many rows or columns as needed. However, only one argument can be 0. In the statement above, the GridLayout object will have as many rows as needed and 2 columns. There will be 10 pixels between columns and 5 pixels between rows. The GridLayout class is part of the java.awt package. Code using this manager requires an `import java.awt.*` statement.

java.awt

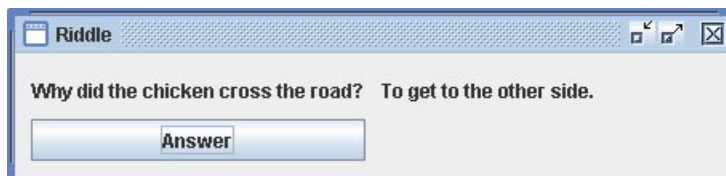
Review: Sunflower

Create a Sunflower application that displays the Latin name for the sunflower when Latin is clicked and the English name when English is clicked. The application GUI should use a BorderLayout manager and look similar to the following after Latin has been clicked:



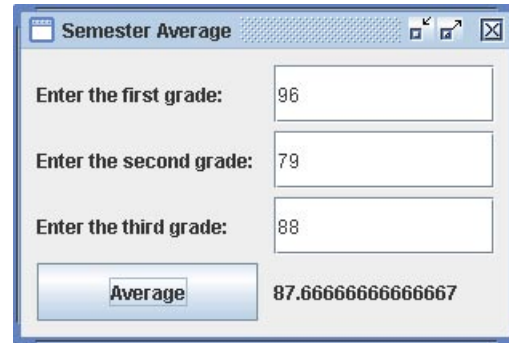
Review: Riddle

Create a Riddle application that displays a riddle and then solves it when Answer is clicked. The application GUI should use a GridLayout manager and look similar to the following after Answer has been clicked:



Getting Input from the User

A *text field* allows a user to enter information at run time. Text fields are usually placed next to a label to prompt the user for the type of data expected in the text field. For example, the SemesterAvg application prompts the user to enter three test grades and then displays the average of the grades when Average is clicked:



The swing package includes the JTextField class with the following constructors and methods:

Class JTextField (javax.swing.JTextField)

Constructor/Methods

JTextField(int col)

creates a JTextField object with width col.

JTextField(String text, int col)

creates a JTextField object displaying default text text in a field with width col.

getText()

returns a String containing the text in the JTextField.

addActionListener(Object)

adds an object that listens for the user to press the Enter key in this component.

converting between text and numeric data

The information typed into a text field is a string, even when numeric data is entered. Conversely, the setText() method of a JLabel expects a string even when the data is numeric. Class methods in the Double and Integer classes can be used to convert data between numeric and String types:

Class Double (java.lang.Double)

Method

parseDouble(String text)

returns the double value in the String text.

toString(double num)

returns the String representation of num.

Class Integer (java.lang.Integer)

Method

parseInt(String text)

returns the int value in the String text.

toString(int num)

returns the String representation of num.

The `parseDouble()` and `parseInteger()` methods are used to convert String data to numeric. The SemesterAvg application, the GUI shown on the previous page, uses `parseDouble()`:

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

public class SemesterAvg implements ActionListener {
    JFrame frame;
    JPanel contentPane;
    JLabel prompt1, prompt2, prompt3, average;
    JTextField grade1, grade2, grade3;
    JButton avgButton;

    public SemesterAvg(){
        /* Create and set up the frame */
        frame = new JFrame("Semester Average");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a GridLayout */
        contentPane = new JPanel();
        contentPane.setLayout(new GridLayout(0, 2, 10, 5));
        contentPane.setBorder(BorderFactory.createEmptyBorder
                                (10, 10, 10, 10));

        /* Create and add a prompt and then a text field */
        prompt1 = new JLabel("Enter the first grade: ");
        contentPane.add(prompt1);

        grade1 = new JTextField(10);
        contentPane.add(grade1);

        /* Create and add a second prompt and
           then a text field */
        prompt2 = new JLabel("Enter the second grade: ");
        contentPane.add(prompt2);

        grade2 = new JTextField(10);
        contentPane.add(grade2);

        /* Create and add a third prompt and then a text field */
        prompt3 = new JLabel("Enter the third grade: ");
        contentPane.add(prompt3);

        grade3 = new JTextField(10);
        contentPane.add(grade3);

        /* Create and add button that will display the
           average of the grades */
        avgButton = new JButton("Average");
        avgButton.setActionCommand("Average");
        avgButton.addActionListener(this);
        contentPane.add(avgButton);

        /* Create and add a label that will display the
           average */
        average = new JLabel(" ");
        contentPane.add(average);

        /* Add content pane to frame */
        frame.setContentPane(contentPane);
    }
}
```

```

/* Size and then display the frame. */
frame.pack();
frame.setVisible(true);
}

/**
 * Handle button click action event
 * pre: none
 * post: The average of the grades entered has been
 * calculated and displayed.
 */
public void actionPerformed(ActionEvent event) {
    String eventName = event.getActionCommand();

    if (eventName.equals("Average")) {
        double avgGrade;
        String g1 = grade1.getText();
        String g2 = grade2.getText();
        String g3 = grade3.getText();

        avgGrade = (Double.parseDouble(g1) + Double.parseDouble(g2)
            + Double.parseDouble(g3))/3;
        average.setText(Double.toString(avgGrade));
    }
}

/**
 * Create and show the GUI.
 */
private static void runGUI() {
    JFrame.setDefaultLookAndFeelDecorated(true);

    SemesterAvg myGrades = new SemesterAvg();
}

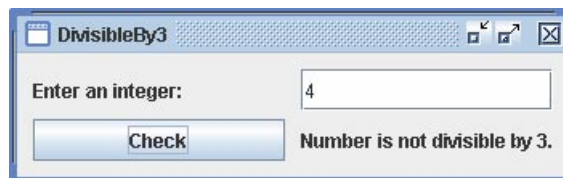
```

...main() method of SemesterAvg code

In the actionPerformed() method, the strings are read from the text fields. To calculate the average grade, each string is parsed for a double value (with parseDouble()). The avgGrade value was then converted to a String with the Double class method toString().

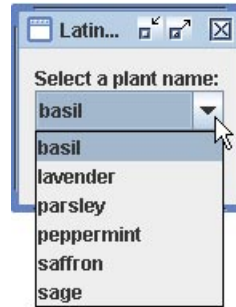
Review: DivisibleBy3

Create a DivisibleBy3 application that prompts the user for an integer and then displays a message when Check is clicked indicating whether the number is divisible by 3. The application interface should look similar to the following after the user has typed a number and clicked Check:



Combo Boxes

A *combo box* offers a user a way to select from a limited set of choices. Combo boxes can offer choices without taking up much room on the interface. The user simply clicks the combo box arrow to display additional choices. The LatinPlantNames application allows the user to select a plant name from a combo box:



The swing package includes the `JComboBox` class for creating combo boxes:

Class `JComboBox` (`javax.swing.JComboBox`)

Methods

`JComboBox(Object[] items)`

creates a `JComboBox` object that contains the items from the `items` array, which must be an array of objects.

`setSelectedIndex(int index)`

makes the item at index `index` the selected item.

`getSelectedItem()`

returns the `String` corresponding to the selected `JComboBox` item.

`setEditable(boolean)`

allows text to be typed in the combo box when `true` is the boolean argument.

`addActionListener(Object)`

adds an object that listens for the user to select an item from this component's list.

JComboBox event

Handling an event from a `JComboBox` requires two lines of code that are similar to:

```
JComboBox comboBox = (JComboBox)event.getSource();
String itemName = (String)comboBox.getSelectedItem();
```

The first statement determines the source of the action event and then the second statement determines which item has been selected.

The LatinPlantNames application code includes a JComboBox object:

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

public class LatinPlantNames implements ActionListener {
    JFrame frame;
    JPanel contentPane;
    JComboBox plantNames;
    JLabel plantListPrompt, latinName;

    public LatinPlantNames(){
        /* Create and set up the frame */
        frame = new JFrame("LatinPlantNames");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a BorderLayout and
           empty borders */
        contentPane = new JPanel();
        contentPane.setLayout(new BorderLayout(contentPane,
                                              BorderLayout.PAGE_AXIS));
        contentPane.setBorder(BorderFactory.createEmptyBorder
                               (10, 10, 10, 10));

        /* Create a combo box and a descriptive label */
        plantListPrompt = new JLabel("Select a plant name: ");
        plantListPrompt.setAlignmentX(JLabel.LEFT_ALIGNMENT);
        contentPane.add(plantListPrompt);

        String[] names = {"basil", "lavender", "parsley",
                          "peppermint", "saffron", "sage"};
        plantNames = new JComboBox(names);
        plantNames.setAlignmentX(JComboBox.LEFT_ALIGNMENT);
        plantNames.setSelectedIndex(0);
        plantNames.addActionListener(this);
        contentPane.add(plantNames);

        /* Create and add a label that will display the
           Latin names */
        latinName = new JLabel("Ocimum");
        latinName.setBorder(BorderFactory.createEmptyBorder
                             (20, 0, 0, 0));
        contentPane.add(latinName);

        /* Add content pane to frame */
        frame.setContentPane(contentPane);

        /* Size and then display the frame. */
        frame.pack();
        frame.setVisible(true);
    }
}
```

```

/**
 * Handle a selection from the combo box
 * pre: none
 * post: The Latin name for the selected plant
 * has been displayed.
 */
public void actionPerformed(ActionEvent event) {
    JComboBox comboBox = (JComboBox)event.getSource();
    String plantName = (String)comboBox.getSelectedItem();

    if (plantName == "basil") {
        latinName.setText("Ocimum");
    } else if (plantName == "lavender") {
        latinName.setText("Lavandula spica");
    } else if (plantName == "parsley") {
        latinName.setText("Apium");
    } else if (plantName == "perppermint") {
        latinName.setText("Mentha piperita");
    } else if (plantName == "saffron") {
        latinName.setText("Crocus");
    } else if (plantName == "sage") {
        latinName.setText("Salvia");
    }
}

```

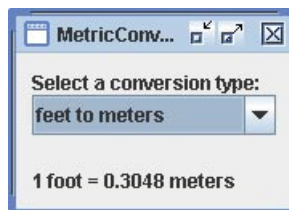
...runGUI() and main() method of LatinPlantNames code

The LatinPlantNames application displays the following output after sage has been selected:



Review: MetricConversion

Create a MetricConversion application that allows the user to select a type of conversion from a combo box and then the corresponding formula is displayed in a label. To convert from the length measurement inches to centimeters, the formula is 1 inch = 2.54 centimeters. The formula 1 foot = 0.3048 meters converts the distance measurement feet to meters. The volume measurement gallon is converted to liters with the formula 1 gallon = 4.5461 liters. The formula 1 pound = 0.4536 kilograms converts the mass measurement pound to kilograms. The application interface should look similar to the following after the user has selected feet to meters:



Changing Colors

Swing components have methods for changing their colors. For example, the content pane can be green, buttons can be magenta, combo boxes can display text in pink. When making color choices, keep the end user in mind because colors can give an application a fun and exciting look or completely turn users away.

Color class

The `java.awt` package includes the `Color` class with numerous color constant members. The following methods, in the Swing component classes, can be used to change the background and foreground colors of the components:

```
setBackground(Color.constant)
```

sets the background color of a component to constant from the `Color` class.

```
setForeground(Color.constant)
```

sets the foreground color of a component to constant from the `Color` class.

How the color change affects the component varies. For a `JLabel`, the foreground color refers to the color of the text. For a `JComboBox`, the foreground color refers to the color of the text in the list.

The `ColorDemo` application demonstrates the color possibilities for components. The application GUI and the code, except for the `runGUI()` and `main()` methods, follow:



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

public class ColorDemo implements ActionListener {
    JFrame frame;
    JPanel contentPane;
    JTextField name;
    JButton displayMessage;
    JLabel textFieldPrompt, hello;

    public ColorDemo(){
        /* Create and set up the frame */
        frame = new JFrame("ColorDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a BorderLayout
           and empty borders */
        contentPane = new JPanel();
        contentPane.setBorder(BorderFactory.createEmptyBorder
                               (10, 10, 10, 10));

        contentPane.setBackground(Color.white);
        contentPane.setLayout(new GridLayout(0, 2, 5, 10));

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

        public class ColorDemo implements ActionListener {
            JFrame frame;
            JPanel contentPane;
            JTextField name;
            JButton displayMessage;
            JLabel textFieldPrompt, hello;

            public ColorDemo(){
                /* Create and set up the frame */
                frame = new JFrame("ColorDemo");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                /* Create a content pane with a BorderLayout
                   and empty borders */
                contentPane = new JPanel();
                contentPane.setBorder(BorderFactory.createEmptyBorder
                                       (10, 10, 10, 10));

                contentPane.setBackground(Color.white);
                contentPane.setLayout(new GridLayout(0, 2, 5, 10));
```

```

/* Create a text field and a descriptive label */
textFieldPrompt = new JLabel("Type your name: ");
textFieldPrompt.setForeground(Color.red);
contentPane.add(textFieldPrompt);

name = new JTextField(10);
name.setBackground(Color.pink);
name.setForeground(Color.darkGray);
contentPane.add(name);

/* Create a Display Message button */
displayMessage = new JButton("Display Message");
displayMessage.setBackground(Color.yellow);
displayMessage.setForeground(Color.blue);
displayMessage.addActionListener(this);
contentPane.add(displayMessage);

/* Create a label that will display a message */
hello = new JLabel(" ");
hello.setForeground(Color.green);
contentPane.add(hello);

/* Add content pane to frame */
frame.setContentPane(contentPane);

/* Size and then display the frame. */
frame.pack();
frame.setVisible(true);
}

/**
 * Handle a the button click
 * pre: none
 * post: A message has been displayed.
 */
public void actionPerformed(ActionEvent event) {
    String text = name.getText();

    hello.setText("Hello " + text);
}

```

...runGUI() and main() method of ColorDemo code

Adding Images

GIF, JPG

Images can make an application more informative, easier to use, and fun. Labels and buttons are often used to display an image, but many Swing components support images. Images that are GIF and JPG format work best. Unless a different path is specified in the application code, the image files must be in the same location as the compiled code.

The JLabel class includes a constructor and a method for displaying images in a label:

Class JLabel (javax.swing.JLabel)

Constructor/Method

JLabel(ImageIcon pic)

creates a JLabel object containing pic.

setIcon(ImageIcon pic)

sets the JLabel to contain pic.

The JButton class includes a constructor and a method for displaying images in a button:

Class JButton (javax.swing.JButton)

Constructor/Method

JButton(String str, ImageIcon pic)

creates a JButton object containing the text `str` and the image `pic`.

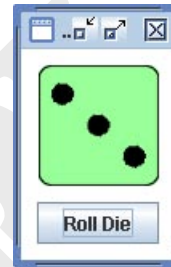
JButton(ImageIcon pic)

creates a JButton object containing the image `pic`.

setIcon(ImageIcon pic)

sets the JButton to contain `pic`.

The Roll application displays a die face. The user can click Roll Die to “roll the die” and display the outcome of the roll. The application GUI and the code, except for the `runGUI()` and `main()` methods, follow:



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

public class Roll implements ActionListener {
    JFrame frame;
    JPanel contentPane;
    JButton rollDie;
    JLabel dieFace;

    public Roll(){
        /* Create and set up the frame */
        frame = new JFrame("Roll");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a BorderLayout and
        empty borders */
        contentPane = new JPanel();
        contentPane.setBorder(BorderFactory.createEmptyBorder
            (10, 10, 10, 10));
        contentPane.setBackground(Color.white);
        contentPane.setLayout(new BorderLayout(contentPane,
            BorderLayout.PAGE_AXIS));

        /* Create a label that shows a die face */
        dieFace = new JLabel(new ImageIcon("die3.gif"));
        dieFace.setAlignmentX(JLabel.CENTER_ALIGNMENT);
        dieFace.setBorder(BorderFactory.createEmptyBorder
            (0, 0, 10, 0));

        contentPane.add(dieFace);
```

```

/* Create a Roll Die button */
rollDie = new JButton("Roll Die");
rollDie.setAlignmentX(JButton.CENTER_ALIGNMENT);
rollDie.addActionListener(this);
contentPane.add(rollDie);

/* Add content pane to frame */
frame.setContentPane(contentPane);

/* Size and then display the frame. */
frame.pack();
frame.setVisible(true);
}

/**
 * Handle a button click
 * pre: none
 * post: A die has been rolled. Matching image shown.
 */
public void actionPerformed(ActionEvent event) {
    Random rand = new Random();
    int newRoll;

    newRoll = rand.nextInt(6) + 1;
    if (newRoll == 1) {
        dieFace.setIcon(new ImageIcon("die1.gif"));
    } else if (newRoll == 2) {
        dieFace.setIcon(new ImageIcon("die2.gif"));
    } else if (newRoll == 3) {
        dieFace.setIcon(new ImageIcon("die3.gif"));
    } else if (newRoll == 4) {
        dieFace.setIcon(new ImageIcon("die4.gif"));
    } else if (newRoll == 5) {
        dieFace.setIcon(new ImageIcon("die5.gif"));
    } else if (newRoll == 6) {
        dieFace.setIcon(new ImageIcon("die6.gif"));
    }
}

```

An image must be an object of the `ImageIcon` class for use in a Swing GUI. The `ImageIcon` class, from the `java.swing` package, has a constructor that accepts a file name as an argument and then converts that file to an `ImageIcon` object.

Review: Roll

Modify the Roll application to roll two dice. Include an image for each die. Change the colors of the components to be more exciting, while still allowing the user to easily read the text on the button and to see the die images. The die images are named `die1.tif`, `die2.tif`, `die3.tif`, `die5.tif`, and `die6.tif` and are supplied as data files for this text.

Using Nested Classes to Handle Events

A GUI can quickly become complex, as a combination of buttons, text fields, and other components that must handle events are added. When a variety of components are on a single interface, separate `actionPerformed()` methods should handle their events.

Up to this point, an instance of the controlling class (`this`) implemented an `actionPerformed()` method. This single method was used to handle events for every component on the GUI. This was sufficient for a simple GUI. However, a GUI with more than one type of component responding to an event should have multiple listeners. One way to implement multiple listeners in a single application is to create each listener from a nested class.

TIP A nested class is also called an inner class.

outer class

A *nested class* is a class within a class. A nested class is a member of the class it is within. As a class member, it has access to all the other members of the class, including private member variables and methods. A class that contains a class member is called an *outer class*. The Semester Stats application uses two nested classes to respond to events:

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

public class SemesterStats {
    JFrame frame;
    JPanel contentPane;
    JLabel prompt1, prompt2, prompt3, stat;
    JTextField grade1, grade2, grade3;
    JButton avgButton, minButton, maxButton;

    public SemesterStats(){
        /* Create and set up the frame */
        frame = new JFrame("Semester Stats");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create content pane with a GridLayout and empty borders */
        contentPane = new JPanel();
        contentPane.setLayout(new GridLayout(0, 2, 10, 5));
        contentPane.setBorder(BorderFactory.createEmptyBorder
            (10, 10, 10, 10));

        /* Create and add a prompt and then a text field */
        prompt1 = new JLabel("Enter the first grade: ");
        contentPane.add(prompt1);

        grade1 = new JTextField(10);
        contentPane.add(grade1);

        /* Create and add a second prompt and then a text field */
        prompt2 = new JLabel("Enter the second grade: ");
        contentPane.add(prompt2);

        grade2 = new JTextField(10);
        contentPane.add(grade2);

        /* Create and add a third prompt and then a text field */
        prompt3 = new JLabel("Enter the third grade: ");
        contentPane.add(prompt3);

        grade3 = new JTextField(10);
        contentPane.add(grade3);
```

```

/* Create and add button that will display the average grade */
avgButton = new JButton("Average");
avgButton.addActionListener(new AvgListener());
contentPane.add(avgButton);

/* Create and add button that will display the min grade */
minButton = new JButton("Min");
minButton.setActionCommand("Min");
minButton.addActionListener(new MinMaxListener());
contentPane.add(minButton);

/* Create and add button that will display the max grade */
maxButton = new JButton("Max");
maxButton.setActionCommand("Max");
maxButton.addActionListener(new MinMaxListener());
contentPane.add(maxButton);

/* Create and add a label that will display stats */
stat = new JLabel(" ");
stat.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));
contentPane.add(stat);

/* Add content pane to frame */
frame.setContentPane(contentPane);

/* Size and then display the frame. */
frame.pack();
frame.setVisible(true);
}

class AvgListener implements ActionListener {
    /**
     * Handle Average button click event
     * pre: none
     * post: The grade average has been calculated and displayed.
     */
    public void actionPerformed(ActionEvent event) {
        double avgGrade;
        String g1 = grade1.getText();
        String g2 = grade2.getText();
        String g3 = grade3.getText();

        avgGrade = (Double.parseDouble(g1) + Double.parseDouble(g2) +
                    Double.parseDouble(g3))/3;
        stat.setText(Double.toString(avgGrade));
    }
}

```

```

class MinMaxListener implements ActionListener {

    /**
     * Handles the Min and Max button click events
     * pre: none
     * post: The minimum or maximum grade has been
     * determined and displayed.
     */
    public void actionPerformed(ActionEvent event) {
        String eventName = event.getActionCommand();
        double minGrade = 999;
        double maxGrade = 0;
        double[] grades = new double[3];

        grades[0] = Double.parseDouble(grade1.getText());
        grades[1] = Double.parseDouble(grade2.getText());
        grades[2] = Double.parseDouble(grade3.getText());

        if (eventName.equals("Min")) {
            for (int i = 0; i < 3; i++) {
                if (minGrade > grades[i]) {
                    minGrade = grades[i];
                }
            }
            stat.setText(Double.toString(minGrade));
        } else if (eventName.equals("Max")) {
            for (int i = 0; i < 3; i++) {
                if (maxGrade < grades[i]) {
                    maxGrade = grades[i];
                }
            }
            stat.setText(Double.toString(maxGrade));
        }
    }
}

/**
 * Create and show the GUI.
 */
private static void runGUI() {
    JFrame.setDefaultLookAndFeelDecorated(true);

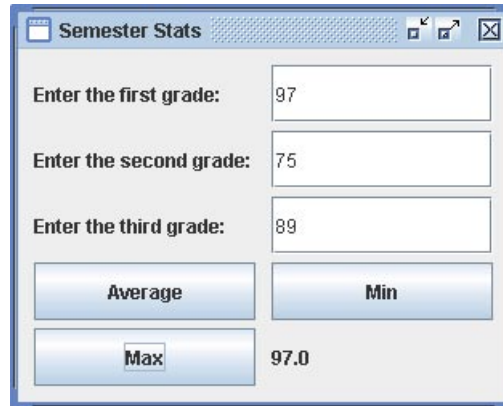
    SemesterStats myGrades = new SemesterStats();
}

public static void main(String[] args) {
    /* Methods that create and show a GUI should be
     run from an event-dispatching thread */
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            runGUI();
        }
    });
}
}

```

One class (AvgListener) was used to implement a listener for the Average button and a second class (MinMaxListener) implements a listener for the Min and Max buttons. These buttons were combined into one listener because their algorithms closely match.

The SemesterStats application displays output similar to:



The screenshot shows a window titled "Semester Stats". It contains three input fields for grades: "Enter the first grade:" with the value 97, "Enter the second grade:" with the value 75, and "Enter the third grade:" with the value 89. Below these fields are three buttons: "Average", "Min", and "Max". The "Max" button is highlighted, and the value 97.0 is displayed next to it.

Chapter 11 Case Study

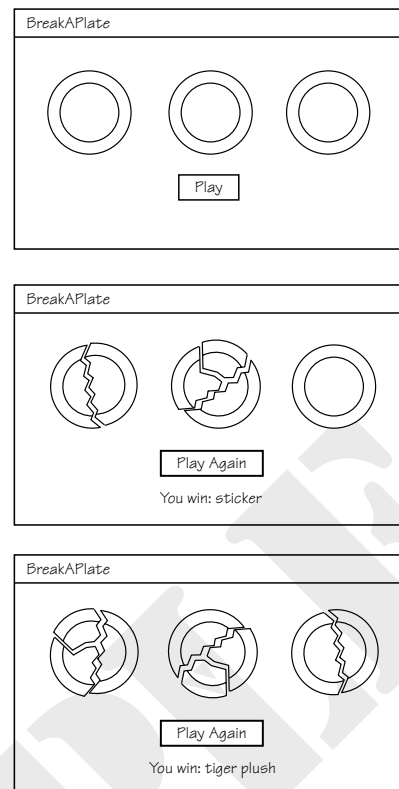
In this case study, a Swing GUI will be created for an application that plays a carnival game called Break-A-Plate. The Break-A-Plate game allows a player to try to break all three plates. If all three plates are broken, a first prize is awarded, otherwise, a consolation prize is awarded.

BreakAPlate Specification

The BreakAPlate application displays three unbroken plates at the start. Clicking Play plays the game, displays broken plates, and displays the prize won. If all three plates are broken, a tiger plush first prize is awarded. If less than three plates are broken, a sticker consolation prize is awarded. At the end of a game the Play button changes to Play Again. Clicking Play Again displays a set of unbroken plates and the button changes back to Play allowing the user to play repeatedly. The application ends when the user closes the window.

The BreakAPlate game uses random numbers to determine if a player has broken all three plates. When three 1s are generated by the random number generator then the application displays three broken plates. If zero, one, or two ones are generated, then the application displays two broken plates indicating a loss.

The BreakAPlate interface should be a GUI that includes a label for displaying the image of the plates, a button for allowing the user to play or play again, and a label for displaying the the prize won. The BreakAPlate GUI should look similar to the following before a game is played, after a losing game has been played, and after a winning game has been played:



The BreakAPlate algorithm:

1. When a game is played, generate three random numbers between 0 and 1.
2. If three ones are generated, display three broken plates and the names of the first prize. Otherwise, display two broken plates and the name of the consolation prize.

BreakAPlate Code Design

The BreakAPlate application simulates a game booth at a carnival. The game can therefore be modeled with a GameBooth object. The GameBooth class was created in Chapter 8 and contains the following methods:

TIP Refer to Chapter 8 for the GameBooth code.

Class GameBooth (Lawrenceville Press)

Constructor/Methods

`GameBooth(int cost, String p1, String p2)`

creates a GameBooth object that charges `cost` amount to play, awards a first prize `p1`, and a consolation prize `p2`. If there is no charge for the game, or the charge is not a consideration, then 0 should be the argument for `cost`.

`start()`

simulates a game that allows the player three plays. If the player succeeds all three times, then the name of the first prize is returned. Otherwise, the name of the consolation prize is returned.

`getCost()`

returns the cost of the game.

The cost of the game is not a factor in this application, so 0 will be the argument for the `cost` parameter in the constructor, and the `getCost()` method will not be needed.

Based on the algorithm and the `GameBooth` class, the `BreakAPlate` `actionPerformed()` method pseudocode looks similar to:

```
String eventName = event.getActionCommand();
String prize;

if (Play) {
    prize = breakAPlate.start();
    if (prize is tiger plush) {
        display all broken plates;
    } else if (prize is sticker) {
        display two broken plates
    }
    display prize won
    change button to Play Again
} else if (Play Again) {
    display unbroken plates
    clear prize won text
    change button to Play
}
```

BreakAPlate Implementation

The `BreakAPlate` implementation involves creating the `BreakAPlate` java file and adding the `GameBooth` java file to the project.

The `BreakAPlate` application is implemented below:

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

public class BreakAPlate implements ActionListener {
    static final String FIRST_PRIZE = "tiger plush";
    static final String CONSOLATION_PRIZE = "sticker";
    JFrame frame;
    JPanel contentPane;
    JButton play;
    JLabel plates, prizeWon;
    GameBooth breakAPlate;

    public BreakAPlate(){
        /* initialize game booth and player */
        breakAPlate = new GameBooth(0, FIRST_PRIZE, CONSOLATION_PRIZE);

        /* Create and set up the frame */
        frame = new JFrame("BreakAPlate");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /* Create a content pane with a BoxLayout and empty borders */
        contentPane = new JPanel();
        contentPane.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
        contentPane.setBackground(Color.white);
        contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
    }
}
```

```

/* Create a label that shows the start of the game */
plates = new JLabel(new ImageIcon("plates.gif"));
plates.setAlignmentX(JLabel.CENTER_ALIGNMENT);
plates.setBorder(BorderFactory.createEmptyBorder(10, 10, 20, 10));
contentPane.add(plates);

/* Create a Play button */
play = new JButton("Play");
play.setActionCommand("Play");
play.setAlignmentX(JButton.CENTER_ALIGNMENT);
play.addActionListener(this);
contentPane.add(play);

/* Create a label that will show prizes won */
prizeWon = new JLabel(" ");
prizeWon.setAlignmentX(JLabel.CENTER_ALIGNMENT);
prizeWon.setBorder(BorderFactory.createEmptyBorder(20, 0, 0, 0));
contentPane.add(prizeWon);

/* Add content pane to frame */
frame.setContentPane(contentPane);

/* Size and then display the frame. */
frame.pack();
frame.setVisible(true);
}

/**
 * Handle the button click
 * pre: none
 * post: The appropriate image and message are displayed.
 */
public void actionPerformed(ActionEvent event) {
    String eventName = event.getActionCommand();
    String prize;

    if (eventName == "Play") {
        prize = breakAPlate.start();
        if (prize.equals(FIRST_PRIZE)) {
            plates.setIcon(new ImageIcon("plates_all_broken.gif"));
        } else if (prize.equals(CONSOLATION_PRIZE)) {
            plates.setIcon(new ImageIcon("plates_two_broken.gif"));
        }
        prizeWon.setText("You win: " + prize);
        play.setText("Play Again");
        play.setActionCommand("Play Again");
    } else if (eventName == "Play Again") {
        plates.setIcon(new ImageIcon("plates.gif"));
        prizeWon.setText(" ");
        play.setText("Play");
        play.setActionCommand("Play");
    }
}
}

```

```

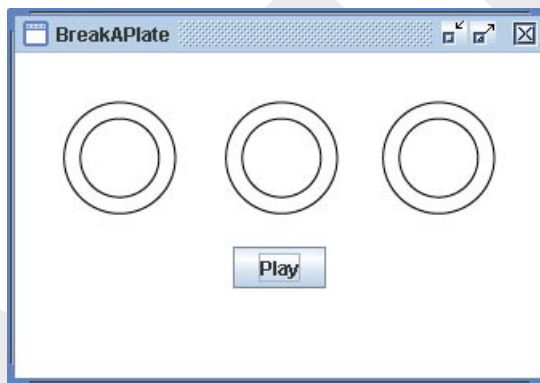
/**
 * Create and show the GUI.
 */
private static void runGUI() {
    JFrame.setDefaultLookAndFeelDecorated(true);

    BreakAPlate carnivalGame = new BreakAPlate();
}

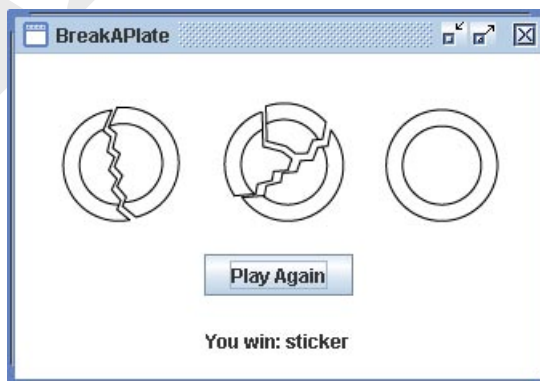
public static void main(String[] args) {
    /* Methods that create and show a GUI should be
     run from an event-dispatching thread */
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            runGUI();
        }
    });
}
}

```

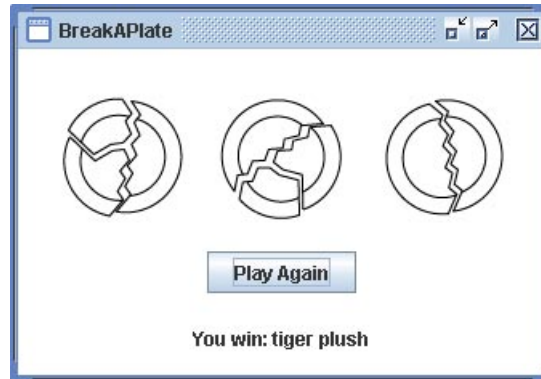
Running the BreakAPlate application displays a GUI similar to:



When Play is clicked, the user either wins or loses. The output below shows a losing game:



The application GUI below shows a winning game:

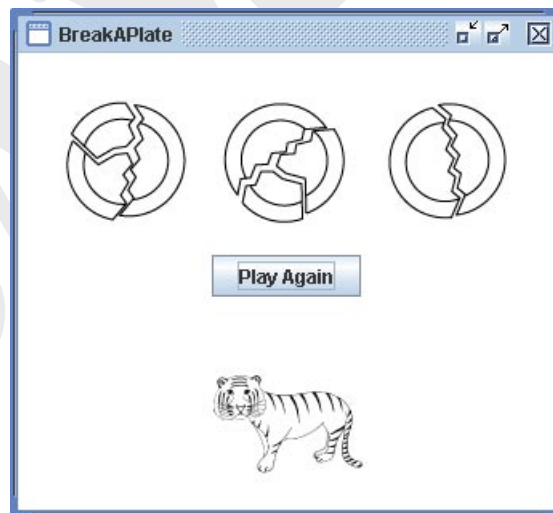


BreakAPlate Testing and Debugging

The application should be tested by playing several games, making sure that the correct prize is named and that the buttons and images change appropriately.

Review: BreakAPlate

Modify the BreakAPlate application to display a picture of the prize won rather than text naming the prize. The `tiger_plush.gif`, `sticker.gif`, and `placeholder.gif` are supplied as data files for this text. The `placeholder.gif` file is a white square that is the same size as the `tiger_plush` and `sticker` images. The `placeholder.gif` file should be displayed in the label at the start of each game. The modified BreakAPlate interface should look similar to the following after the user has played a winning game:



Chapter Summary

This chapter introduced graphical user interfaces that use the Swing API. The Java swing package contains many component classes, including JFrames, JPanels, JLabels, JButtons, JComboBoxes, and JTextFields.

Applications with a GUI are event-driven. The GUI must be run from an event-dispatching thread. A GUI application responds to interactions from the user called events. An event executes a method called an event handler. A Swing event handler is also called an action event handler.

A frame is a container for content panels. A commonly used content panel is the JPanel component. A JPanel can contain other components such as labels and buttons. A label is a component that does not receive events. A button is a commonly used component for accepting input from the user.

Only components with a listener object can execute an event handler. A listener executes the actionPerformed() event handler, passing it the an ActionEvent object that contains the action command describing the event.

The layout of components in a JPanel can be controlled with layout manager and the use of borders and alignment. The FlowLayout, BorderLayout, and GridLayout are the three managers covered in this chapter. The placement of components within a layout can be controlled further with empty borders for padding and using alignment.

The text field component allows a user to type information into the interface. The information typed into a text field is a string. The Integer and Double classes include class methods for converting information to numeric data. The label component requires a string when setting the text. The Integer and Double classes also provide the toString() method for converting numeric data to a String.

Combo boxes allow the user to select from a limited set of choices. A combo box does not take up much space and displays a list of choices when its arrow button is clicked.

Most of the components include methods for setting the background and foreground colors. Color can make an application easier to use and more interesting, but should be selected with the user in mind.

GIF and JPG images can be included in an application GUI. A label or a button is commonly used to display an image.

Nested classes are used to implement multiple event handlers. Nested classes, like any other member of a class, have access to other members of the class, including private variables and method members. A class that contain a class member is called an outer class.

Vocabulary

Action command A string describing an event.

Alignment The placement of components within a layout.

BoxLayout manager A layout manager that places components one after the other in a column.

Button A GUI component that the user can click.

Combo box A GUI component that offers a user a way to select from a limited set of choices.

Container A component that holds and displays all the other components of a GUI.

Content frame A top-level container.

Event A user interaction with an application's GUI.

Event-driven application An application that responds to events.

Event handler A method that executes in response to an event.

FlowLayout manager A layout manager that places components one next to the other in a row.

Frame A GUI window that contains a border, title, and maximize, minimize, and close buttons.

GridLayout manager A layout manager that places components in a grid of rows and columns.

Label A GUI component that displays text or an image and does not interact with the user.

Layout The arrangement of components.

Layout manager Used to specify the order of components on a content pane.

Listener An object that listens for action events.

Nested class A class that is a member of another class. A class within a class.

Outer class A class that contains a class member.

Text field A GUI component that allows a user to enter information at run time.

Thread A process that runs sequentially from start to finish.

ActionListener The `java.awt.event` interface that contains the `actionListener()` method.

actionPerformed() A method that is implemented to respond to events. This method is the only method in the `ActionListener` interface.

BorderFactory The `java.lang` class with class methods for creating a border object.

BoxLayout A `javax.swing` class for setting the layout of a content pane.

Color The `java.awt` class that contains constants for changing component colors.

Double A `java.lang` class for converting numbers between numeric and text data.

GridLayout A `java.awt` class for setting the layout of a content pane.

ImageIcon A `java.swing` class for converting an image, such as a GIF or JPG, to an `ImageIcon` object.

Integer A `java.lang` class for converting numbers between numeric and text data.

java.awt The package containing the `GridLayout` class and `Color` class constants.

java.awt.event The package containing the `ActionListener` interface.

java.swing The package containing the `Swing` API classes.

JButton A `java.swing` class for creating a button in a GUI. The class includes methods for adding text or images to the label.

JComboBox A `java.swing` class for creating a combo box in a GUI.

JFrame A `java.swing` class for creating a window, also called a frame, in a GUI.

JLabel The `java.swing` class for creating a label in a GUI. The class includes methods for adding text or images to the label.

JPanel The `java.swing` class for creating a content panel for a frame. The class includes methods for adding and removing components.

JTextField The `java.swing` class for creating a text field.

FlowLayout A `javax.swing` class for setting the layout of a content pane.

Swing API A part of the Java Foundation Classes that contains numerous components for creating GUIs.

this The keyword for indicating an object itself.

Critical Thinking

1. Explain the difference between an event-driven application and a console-based application.
2. Explain how code is executed in an event-driven application.
3. Can components be added directly to a frame? Explain.
4. Can a label respond to events? Explain.
5. Why do you think a GUI needs to be run from an event-dispatching thread?
6. What is the difference between a label and a button?
7. a) What does `this` indicate when used as the argument for the `addActionListener()` method?
b) What must the object listening for an event contain?
8. List three ways to control the layout of a content pane.
9. What type of borders are used to add padding around components?
10. List three layout managers and explain how each arranges the components on a content pane.
11. Which class is used to return an object for the `setBorder()` method?
12. Are borders for padding necessary when the `GridLayout` manager is used? Explain.
13. What must first be done with numeric data typed in a text field before it can be used in a calculation?
14. What is the value of `num1` in the last statement below?
15. An application prompts a user to select a name from a list of six names. Which is a better component choice: a text field or a combo box? Explain.
16. Would white text on a pink background be a good color combination for a GUI? Why or why not?
17. What image types are supported in a Swing GUI?
18. Which components are often used to display an image?
19. What must an image be converted to in order to be displayed by a Swing component?

True/False

20. Determine if each of the following are true or false. If false, explain why.
 - a) `JFrame` is a class in the `javax.swing` package.
 - b) A button click is an event handler.
 - c) Labels can be changed by the user.
 - d) A `JFrame` object uses a content pane to hold GUI components.
 - e) The index value for the first component on a content pane is 1.
 - f) The `JLabel` class constant `LEADING` indicates right alignment.
 - g) A thread is a sequential process that runs from start to finish.
 - h) Swing components use listeners to determine if an event has occurred.
 - i) A class that uses a listener must implement an `ActionListener` class.
 - j) An empty border indicates that there is no space around a component.
 - k) The `FlowLayout` manager places components one after the other in a column, with one column per line.
 - l) All components on a GUI must have the same alignment.
 - m) The class constant `PAGE_AXIS` specifies that components should be arranged in a vertical line.
 - n) The `JLabel` `setAlignmentX()` method is used to specify the horizontal alignment of the components within the layout.

```
double num1;  
Double num2 = new Double(3);  
String num3 = "5";  
num1 = num2.doubleValue() +  
    Double.valueOf(num3).doubleValue();
```

- o) Code using the BorderLayout manager requires an `import java.awt.*` statement.
- p) Information is entered into a text field at run time.
- q) A text field can only accept numeric data.
- r) The JLabel foreground color refers to the text color.
- s) A Swing GUI can have only one listener.
- t) A nested class has access to the `private` variable members of the outer class.

SAMPLE

Exercises

Exercise 1 LocalBankGUI

Create LocalBankGUI application that implements a GUI for the Local Bank case study in Chapter 9.

Exercise 2 TicTacToeGUI

Create a TicTacToeGUI application that allows two players to play a computerized tic-tac-toe game. Refer to the TicTacToe application presented in Chapter 9. The TTT class code will need to be modified for the GUI version of the application. Use a button for each “box” in the tic-tac-toe board.

Exercise 3 PhotoAlbum

Create a PhotoAlbum application that displays a new picture each time Next is clicked. Use the grayangel.jpg, scorpionfish.jpg, sponges.jpg, and starfish.jpg files supplied as data files for this text. The application should allow the user to continuously cycle through the images.

Exercise 4 Clacker

In the game Clacker, the numbers 1 through 12 are initially displayed. The player throws two dice and may cover the number representing the total or the two numbers on the dice. For example, for a throw of 3 and 5, the player may cover the 3 and the 5 or just the 8. Play continues until all the numbers are covered. The goal is to cover all the numbers in the fewest rolls.

Create a Clacker application that displays 12 buttons numbered 1 through 12. Each of these buttons can be clicked to “cover” it. A covered button displays nothing. Another button labeled Roll can be clicked to roll the dice. Include labels to display the appropriate die images for each roll. Another label displays the number of rolls taken. A New Game button can be clicked to clear the labels and uncover the buttons for a new game.

Exercise 5 LifeGUI

Create a LifeGUI application that is based on the Life application created in Chapter 10, Exercise 14. The 20 × 20 grid should be buttons that initially display all 0s. To select the live cells for the first day, the user clicks the buttons in the positions of live cells. When clicked, a button changes from displaying a 0 to displaying a X. A Next button below the grid can be clicked repeatedly to display the next generations until the user quits or there are no more live cells.

SAMPLE