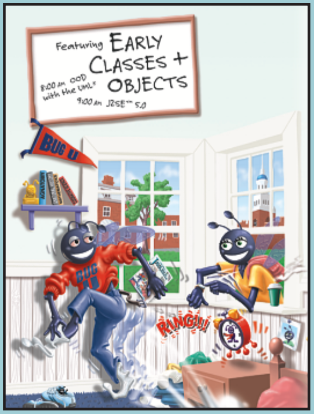
11

GUI Components: Part 1



Do you think I can listen all day to such stuff?
—Lewis Carroll

Even a minor event in the life of a child is an event of that child's world and thus a world event.
—Gaston Bachelard

You pays your money and you takes your choice.
—Punch

Guess if you can, choose if you dare.
—Pierre Corneille

OBJECTIVES

In this chapter you will learn:

- The design principles of graphical user interfaces (GUIs).
- To build GUIs and handle events generated by user interactions with GUIs.
- To understand the packages containing GUI components, event-handling classes and interfaces.
- To create and manipulate buttons, labels, lists, text fields and panels.
- To handle mouse events and keyboard events.
- To use layout managers to arrange GUI components

Assignment Checklist

Name: ______________________ **Date:** ______________________

Section: ______________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	19, 20, 21, 22, 23, 24, 25, 26, 27, 28	
Short Answer	29, 30, 31, 32, 33, 34	
Programming Output	35, 36, 37, 38, 39	
Correct the Code	40, 41, 42	
Lab Exercises		
Exercise 1 — Guess Game	YES NO	
Follow-Up Questions and Activities	1, 2	
Exercise 2 — Events	YES NO	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
PostLab Activities		
Coding Exercises	1, 2, 3, 4	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: ____________________ **Date:** ____________________

Section: ____________________

After reading Chapter 11 of *Java How to Program: Sixth Edition*, answer the given questions. The questions are intended to reinforce your understanding of key concepts. You may answer the questions before or during the lab.

For each term in the left column, write the letter for the description from the right column that best matches the term.

	Term	Description
___	1. `ImageIcon`	a) A single-line area in which text can be entered by the user, but the text is hidden automatically.
___	2. `ItemListener`	b) Handles key events that are generated when keys on the keyboard are pressed and released.
___	3. `JLabel`	c) Displays a series of items from which the user may select one or more items.
___	4. `KeyEvent`	d) Is used to load images of various formats, including Portable Network Graphics (PNG) format.
___	5. `JList`	e) Layout manager that arranges components into five regions: North, South, East, West, and Center.
___	6. `KeyListener`	f) Provides scrolling capabilities for a component.
___	7. `JRadioButton`	g) Objects of subclasses of this type can have rollover icons that appear when the mouse moves over such components in a GUI.
___	8. `ActionEvent`	h) Provides text instructions or information on a GUI.
___	9. `JScrollPane`	i) Allows a `JList` user to select multiple items and those items are not required to be contiguous.
___	10. `AbstractButton`	j) Must define method `itemStateChanged`.
___	11. interface `SwingConstants`	k) Layout manager that divides the container into a grid of rows and columns.
___	12. `JComponent`	l) A `JButton` generates this event type when the user presses the button.
___	13. `SINGLE_INTERVAL_SELECTION`	m) Superclass to most Swing components.
___	14. `JPasswordField`	n) Generates an `ItemEvent` when clicked.
___	15. `GridLayout`	o) Layout manager that lays out components left to right in the order in which they are added to the container.
___	16. `MULTIPLE_INTERVAL_SELECTION`	p) Allows `JList` user to select contiguous items.
___	17. `BorderLayout`	q) Maintains a set of virtual key-code constants that represent every key on the keyboard.
___	18. `FlowLayout`	r) Defines a set of common integer constants that are used with many Swing components.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________ **Date:** ____________________

Section: ____________________

Fill in the blanks for each of the following statements:

19. `JPasswordField` method `getPassword` returns the password as a(n) ________.

20. A(n) ________ manages the relationship between several `JRadioButton`s.

21. Class ________ provides prepackaged dialog boxes for both input and output.

22. Swing GUI components are defined in package ________.

23. The Swing GUI components contain three state button types: ________, ________ and ________.

24. Method ________ specifies whether the user can modify the text in a `JTextComponent`.

25. `JComponent` method ________ specifies the tooltip that is displayed when the user positions the mouse over a lightweight GUI component.

26. `JFrame` method ________ specifies what should happen when the user closes a `JFrame`.

27. `Container` method ________ recomputes the container's layout using the current layout manager for the `Container` and the current set of displayed GUI components.

28. When the user types data into a `JTextField` or `JPasswordField` and presses the *Enter* key, an event of type ________ occurs.

Prelab Activities

Name:

Short Answer

Name:	____________________	**Date:**	____________________
Section:	____________________		

Answer the following questions in the space provided. Your answers should be as concise as possible; aim for two or three sentences.

29. What happens if you do not add a GUI component to a container?

30. What happens if you forget to register an event handler for a GUI component?

31. What happens when adding a component to a `BorderLayout` if you do not specify the region in which the component should be placed?

Prelab Activities

Name:

Short Answer

32. What happens when more than one component is added to a particular region in a `BorderLayout`?

33. What happens at execution time if an attempt is made to add a component to a container, but that component has not yet been instantiated?

34. How is an anonymous inner class different from other inner classes?

Prelab Activities

Name:

Programming Output

Name: ____________________ **Date:** ____________________

Section: ____________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.] For the following exercises, draw an approcximate representation of the GUI that appears when the program executes.

35. What does the GUI look like in the following application?

```java
// ProgrammingOutput.java
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class ProgrammingOutput extends JFrame
{
   private JButton cancelJButton;
   private JButton okJButton;
   private JTextField inputJTextField;
   private JLabel nameJLabel;
   private JCheckBox firstNameJCheckBox;
   private JCheckBox lastNameJCheckBox;
   private JPanel checkJPanel;
   private JPanel buttonJPanel;

   // constructor sets up GUI
   public ProgrammingOutput()
   {
      super( "Input Name" );

      // build nameJPanel
      nameJLabel = new JLabel( "Type your name" );
      inputJTextField = new JTextField( 20 );
      setLayout( new FlowLayout() );
      add( nameJLabel );
      add( inputJTextField );

   } // end ProgrammingOutput constructor
} // end class ProgrammingOutput
```

Prelab Activities

Name:

Programming Output

```java
// ProgrammingOutputTest.java
import java.awt.FlowLayout;
import javax.swing.JFrame;

public class ProgrammingOutputTest
{
   // execute application
   public static void main( String args[] )
   {
      ProgrammingOutput application = new ProgrammingOutput();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.setSize( 400, 150 );
      application.setVisible( true );
   } // end main
} // end class ProgrammingOutputTest
```

Your answer:

36. What does the GUI look like after adding the following code segment is added at the end class `ProgrammingOuput`'s constructor in *Programming Output Exercise 35*?

```java
// build checkJPanel
firstNameJCheckBox = new JCheckBox( "First name" );
lastNameJCheckBox = new JCheckBox( "Last name" );
checkJPanel = new JPanel();
checkJPanel.setLayout( new GridLayout( 1 , 2 ) );
checkJPanel.add( firstNameJCheckBox );
checkJPanel.add( lastNameJCheckBox );
add( checkJPanel );

```

Your answer:

Prelab Activities

Name:

Programming Output

37. What does the GUI look like after adding the following code segment is added at the end class `ProgrammingOuput`'s constructor in *Programming Output Exercises 35–36*?

```java
// build buttonJPanel
okJButton = new JButton( "Ok" );
cancelJButton = new JButton( "Cancel" );
buttonJPanel = new JPanel();
buttonJPanel.setLayout( new GridLayout( 1, 2 ) );
buttonJPanel.add( okJButton );
buttonJPanel.add( cancelJButton );
add( buttonJPanel );

```

Your answer:

38. What does the GUI from *Programming Output Exercises 35–37* look like if the following line of code is inserted after line 10 of `ProgrammingOutputTest.java`?

```java
application.setLayout( new FlowLayout( FlowLayout.LEFT, 10, 5 ) );
```

Your answer:

Prelab Activities

Name:

Programming Output

39. What does the GUI from *Programming Output Exercises 35–37* look like if the following line of code is replaces the line of code added to `ProgrammingOutputTest.java` in Programming Output Exercise 38?

```
application.setLayout( new FlowLayout( FlowLayout.RIGHT, 10, 5 ) );
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: _______________ Date: _______________

Section: _______________

Determine if there is an error in each of the following program segments. If there is an error, specify whether it is a logic error or a compilation error, circle the error in the program and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* There may be more than one error in each program segment.]

Assume the following template definition of classes `CorrectTheCode` and `CorrectTheCodeTest`. Note that all the code in *Correct the Code Exercises 40–42* should be placed starting at line 20 in the `CorrectTheCode` constructor.

```
 1  // CorrectTheCode.java
 2  import java.awt.BorderLayout;
 3  import java.awt.event.ActionEvent;
 4  import java.awt.event.ActionListener;
 5
 6  import javax.swing.JButton;
 7  import javax.swing.JFrame;
 8  import javax.swing.JTextArea;
 9
10  public class CorrectTheCode extends JFrame
11  {
12     private JButton okJButton;
13     private JButton clearJButton;
14     private JTextArea contentJTextArea;
15
16     public CorrectTheCode()
17     {
18        super( "CorrectTheCode" );
19
20        /* all the code segments below will be inserted here */
21     } // end CorrectTheCode constructor
22  } // end class CorrectTheCode
```

```
 1  // CorrectTheCodeTest.java
 2  import javax.swing.JFrame;
 3
 4  public class CorrectTheCodeTest
 5  {
 6     // execute application
 7     public static void main( String args[] )
 8     {
 9        CorrectTheCode application = new CorrectTheCode();
10        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        application.setSize( 200, 200 );
12        application.setVisible( true );
13     } // end main
14  } // end class CorrectTheCodeTest
```

Prelab Activities

Name:

Correct the Code

40. The following code should create a JButton with the value OK and add it to the JFrame.

```
okJButton = new JButton();
add( BorderLayout.CENTER );
```

Your answer:

41. The following code segment should create a JButton with the value Clear and a JTextArea in which the user is not allowed to type. The code segment should add these components to the JFrame.

```
clearJButton = new JButton();
add( BorderLayout.SOUTH );

contentJTextArea = new JTextArea( "Type or Click", 1, 4 );
contentJTextArea.setEditable();
add( BorderLayout.NORTH );
```

Your answer:

Prelab Activities

Name:

Correct the Code

42. The following code should add `ActionListeners` to the **OK** and **Clear** buttons defined in *Correct the Code Exercises 40–41* and specify how to handle each button's event with an anonymous inner class.

```
okayJButton.new ActionListener()
{
   public void actionPerformed( Actionevent e )
   {
      contentJTextArea.setText( "You clicked okay: " );
   }
}

clearJButton.new ActionListener()
{
   public void actionPerformed( Actionevent e )
   {
      contentJTextArea.setText( "" );
   }
}
```

Your answer:

Lab Exercises

Lab Exercise 1 — Guess Game

Name: ______________________ **Date:** ______________________

Section: ______________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of Problem
3. Sample Output
4. Program Template (Fig. L 11.1 and Fig. L 11.2)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working Java program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with Java code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 11 of *Java How to Program: Sixth Edition*. In this lab, you will practice:

- Designing a GUI.
- Processing events.
- Creating and manipulating GUI components.

The follow-up questions and activities also will give you practice:

- Using various GUI methods to manipulate components.
- Adding additional components to a GUI.

Problem Description

Write an application that plays "guess the number" as follows: Your application chooses the number to be guessed by selecting an integer at random in the range 1–1000. The application then displays the following in a label:

```
I have a number between 1 and 1000. Can you guess my number?
Please enter your first guess.
```

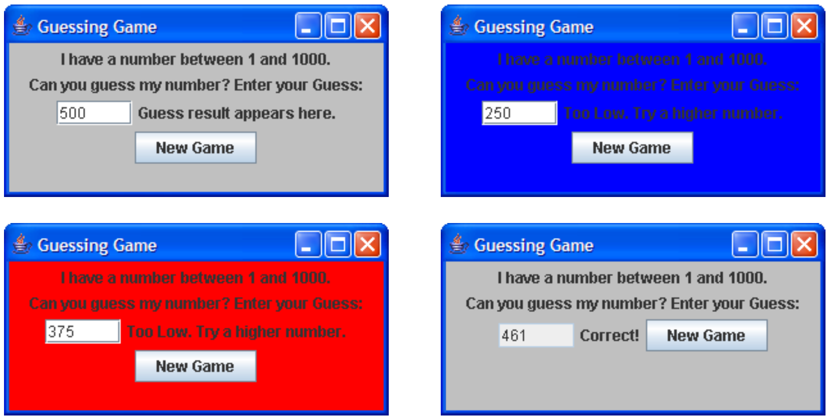
A `JTextField` should be used to input the guess. As each guess is input, the background color should change to either red or blue. Red indicates that the user is getting "warmer," and blue indicates that the user is getting "colder." A `JLabel` should display either `"Too High"` or `"Too Low"` to help the user zero in on the correct answer. When the user gets the correct answer, `"Correct!"` should be displayed, and the `JTextField` used for input should be changed to be uneditable. A `JButton` should be provided to allow the user to play the game again. When the `JButton` is clicked, a new random number should be generated and the input `JTextField` changed to be editable.

Lab Exercises

Name:

Lab Exercise 1 — Guess Game

Sample Output



Guessing Game
I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
500 Guess result appears here.
New Game

Guessing Game
I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
250 Too Low. Try a higher number.
New Game

Guessing Game
I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
375 Too Low. Try a higher number.
New Game

Guessing Game
I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
461 Correct! New Game

Program Template

```java
// Exercise 11.15 Solution: GuessGameFrame.java
// Guess the number
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JButton;

public class GuessGameFrame extends JFrame
{
   private static Random generator = new Random();
   private int number; // number chosen by application
   private int guessCount; // number of guesses
   private int lastDistance; // distance between last guess and number
   private JTextField guessInputJTextField; // for guessing
   private JLabel prompt1JLabel; // first prompt to user
   private JLabel prompt2JLabel; // second prompt to user
   private JLabel messageJLabel; // displays message of game status
   private JButton newGameJButton; // creates new game
   private Color background; // background color of application

   // set up GUI and initialize values
   public GuessGameFrame()
   {
      /* Write a line of code that calls the superclass constructor and sets the title
         of this application to "Guessing Game" */

      guessCount = 0; // initialize number of guesses to 0
      background = Color.LIGHT_GRAY; // set background to light gray

```

Fig. L 11.1 | GuessGameFrame.java. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Guess Game

```java
      prompt1JLabel = new JLabel(
         "I have a number between 1 and 1000." ); // describe game
     prompt2JLabel = new JLabel(
         "Can you guess my number? Enter your Guess:" ); // prompt user

      guessInputJTextField = new JTextField( 5 ); // to enter guesses
      guessInputJTextField.addActionListener( new GuessHandler( ) );
      messageJLabel = new JLabel( "Guess result appears here." );

      /* Write a statement that creaters the "New Game" button */
      newGameJButton.addActionListener(

         new ActionListener() // anonymous inner class
         {
            public void actionPerformed( ActionEvent e )
            {
               /* Write code that resets the application to an appropriate state
                  to start a new game. Reset the background color to light gray,
                  set the JTextFields to their initial text, call method
                  theGame and repaint the GuessGame JFrame */
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      /* Write code that will set the layout of the container to a Flowlayout,
         then add all the GUI components to the container */
     theGame(); // start new game
   } // end GuessGameFrame constructor

   // choose a new random number
   public void theGame()
   {
      /* Write a statement that sets instance variable number to a random number
         between 1 and 1000 */
   } // end method theGame

   // change background color
   public void paint( Graphics g )
   {
      super.paint( g );
      getContentPane().setBackground( background ); // set background
   } // end method paint

   // react to new guess
   public void react( int guess )
   {
      guessCount++; // increment guesses
      /* Write code that sets instance variable currentDistance to 1000. This
         variable's value will be used to determine if th ebackground color
         should be set to red or blue to indicate that the last guess was getting
         closer to or further from the actual number. */

      // first guess
      if ( guessCount == 1 )
      {
```

Fig. L 11.1 | GuessGameFrame.java. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Guess Game

```java
         /* Write code to set instance variable lastDistance to the absolute value
            of the difference between variables guess and number. This value will
            be used with subsequent guesses to help set the background color. */

         if ( guess > number )
            messageJLabel.setText( "Too High. Try a lower number." );
         else
            messageJLabel.setText( "Too Low. Try a higher number." );
      } // end if
      else
      {
         /* Write code that sets instance variable currentDistance to the absolute
            value of the difference between variables guess and number. This
            variable's value will be compared with lastDistance to determine the
            background color. */

         // guess is too high
         if ( guess > number )
         {
            messageJLabel.setText( "Too High. Try a lower number." );

            /* Write code that sets Color variable background to red if the
               currentDistance is less than or equal to lastDistance; otherwise,
               set background to blue. Then assign currentDistance to lastDistance. */
         } // end if
         else if ( guess < number ) // guess is too low
         {
            messageJLabel.setText( "Too Low. Try a higher number." );
            background = ( currentDistance <= lastDistance ) ?
               Color.RED : Color.BLUE;
            lastDistance = currentDistance;
         } // end else if
         else // guess is correct
         {
            messageJLabel.setText( "Correct!" );

            /* Write code that sets Color variable background to red if the
               currentDistance is less than or equal to lastDistance; otherwise,
               set background to blue. Then assign currentDistance to lastDistance. */
         } // end else

         repaint();
      } // end else
   } // end method react

   // inner class acts on user input
   class GuessHandler implements ActionListener
   {
      public void actionPerformed( ActionEvent e )
      {
         /* Write code that will obtain the guess, convert it to an int and
            pass that value to the react method */
      } // end method actionPerformed
   } // end inner class GuessHandler
} // end class GuessGameFrame
```

Fig. L 11.1 | GuessGameFrame.java. (Part 3 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Guess Game

```java
// Exercise 11.15 Solution: GuessGame.java
// Guess the number
import javax.swing.JFrame;

public class GuessGame
{
   public static void main( String args[] )
   {
      GuessGameFrame guessGameFrame = new GuessGameFrame();
      guessGameFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      guessGameFrame.setSize( 300, 150 ); // set frame size
      guessGameFrame.setVisible( true ); // display frame
   } // end main
} // end class GuessGame
```

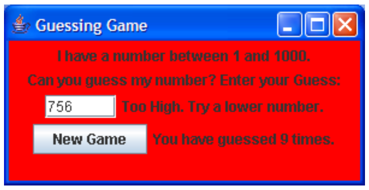
Fig. L 11.2 | `GuessGame.java`.

Problem-Solving Tips

1. Use methods from the `JTextField` class to manipulate all `JTextField` components. For instance, method `setText` will set the text of the text field, and method `setEditable` will set whether the text field can be edited or not.
2. Method `setBackground` from class `JFrame` sets the background color of the `JFrame`.
3. Use method `nextInt` from class `Random` to generate a random number from 1 to 1000. You will need to scale the range of values produced by `random` by 1000 and shift the range by 1.
4. Use variables `lastDistance` and `currentDistance` to determine the distance of the guess from the actual number. If this distance gets larger between guesses, set the background color of the `JFrame` to blue. If this distance gets smaller or stays the same, set the background color to red.
5. If you have any questions as you proceed, ask your lab instructor for assistance.

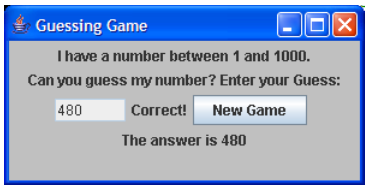
Follow-Up Questions and Activities

1. Modify the previous program to keep track of how many guesses the user has made, and display that number in another `JLabel` in the `JFrame`.



I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
756 Too High. Try a lower number.
You have guessed 9 times.

2. Now modify the previous program so that there is another `JLabel` in the `JFrame` that contains the number to be guessed, but does not become visible, until the user guesses the right number. In other words the `JLabel` is always there, the user just can't see it until the correct number is guessed. [*Hint:* use method `setVisible` to show and hide the `JLabel`.]



I have a number between 1 and 1000.
Can you guess my number? Enter your Guess:
480 Correct!
The answer is 480

Lab Exercises

Name:

Lab Exercise 2 — Events

Name: ____________________ **Date:** ____________________

Section: ____________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of Problem
3. Sample Output
4. Program Template (Fig. L 11.3 and Fig. L 11.4)
5. Problem-Solving Tips

The program template represents a complete working Java program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with Java code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 11 of *Java How to Program: Sixth Edition*. In this lab you will practice:

- Understanding when events occur and how they are generated.
- Displaying information about different events.

Problem Description

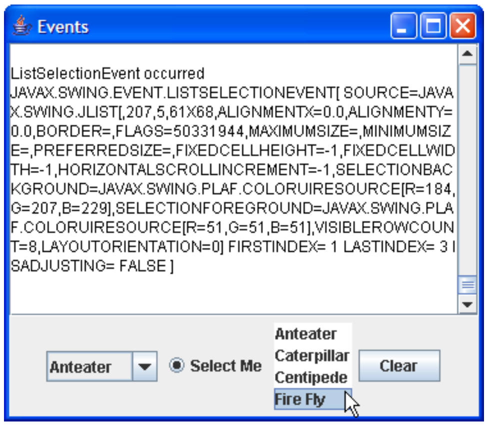
It is often useful to display the events that occur during the execution of an application. This can help you understand when the events occur and how they are generated. Write an application that enables the user to generate and process every event discussed in this chapter. The application should provide methods from the `ActionListener`, `ItemListener`, `ListSelectionListener`, `MouseListener`, `MouseMotionListener` and `KeyListener` interfaces to display messages when the events occur. Use method `toString` to convert the event objects received in each event handler into a `String` that can be displayed. Method `toString` creates a `String` containing all the information in the event object.

Lab Exercises

Name:

Lab Exercise 2 — Events

Sample Output



```
ListSelectionEvent occurred
JAVAX.SWING.EVENT.LISTSELECTIONEVENT[ SOURCE=JAVA
X.SWING.JLIST[,207,5,61X68,ALIGNMENTX=0.0,ALIGNMENTY=
0.0,BORDER=,FLAGS=50331944,MAXIMUMSIZE=,MINIMUMSIZ
E=,PREFERREDSIZE=,FIXEDCELLHEIGHT=-1,FIXEDCELLWID
TH=-1,HORIZONTALSCROLLINCREMENT=-1,SELECTIONBAC
KGROUND=JAVAX.SWING.PLAF.COLORUIRESOURCE[R=184,
G=207,B=229],SELECTIONFOREGROUND=JAVAX.SWING.PLA
F.COLORUIRESOURCE[R=51,G=51,B=51],VISIBLEROWCOUN
T=8,LAYOUTORIENTATION=0] FIRSTINDEX= 1 LASTINDEX= 3 I
SADJUSTING= FALSE ]
```

Program Template

```java
// Exercise 11.16 Solution: EventsFrame.java
// Program displays events that occur during execution.
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JComboBox;
import javax.swing.JRadioButton;
import javax.swing.JList;
import javax.swing.JButton;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;

public class EventsFrame extends JFrame implements ActionListener,
   ItemListener, MouseListener, MouseMotionListener,
   KeyListener, ListSelectionListener
{
   private JPanel panel1;
   private JScrollPane scrollPane;
   private JTextArea outputJTextArea;
   private JComboBox comboBox;
   private JRadioButton radioButton;
   private JList list;
   private JButton clearJButton;
```

Fig. L 11.3 | `EventsFrame.java`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Events

```java

   private String names[] = {
      "Anteater", "Caterpillar", "Centipede", "Fire Fly" };

   // set up GUI and register event handlers
   public EventsFrame()
   {
      super( "Events" );

      // create GUI components
      outputJTextArea = new JTextArea( 10, 30 );
      outputJTextArea.setLineWrap( true );
      outputJTextArea.setEditable( false );
      outputJTextArea.setBackground( Color.WHITE );
      outputJTextArea.setForeground( Color.BLACK );

      // add the output area to a scroll pane
      // so the user can scroll the output
      /* Write a statement that attaches the output JTextArea to a JScrollPane */

      // comboBox listens for item and key events
      comboBox = new JComboBox( names );
      /* Write a statement that registers an ItemListener for this JComboBox */
      /* Write a statement that registers a KeyListener for this JComboBox */

      // radioButton listens for action events
      radioButton = new JRadioButton( "Select Me", false );
      /* Write a statement that registers an ActionListener for
         this JRadioButton */

      // list listens for list selection events
      list = new JList( names );
      list.addListSelectionListener( this );

      // clear button for clearing the output area
      clearJButton = new JButton( "Clear" );
      clearJButton.addActionListener(
         /* Write code that defines an anonymous inner class that
            will clear the output JTextArea when the clear button is clicked */
      ); // end call to addActionListener

      // application listens to its own key and mouse events
      /* Write code that registers a MouseListener
         and a MouseMotionListener for the Events JFrame */

      panel1 = new JPanel();
      panel1.add( comboBox );
      panel1.add( radioButton );
      panel1.add( list );
      panel1.add( clearJButton );

      // add components to container
      setLayout( new BorderLayout() );
      add( scrollPane, BorderLayout.CENTER );
      add( panel1, BorderLayout.SOUTH );
   } // end EventsFrame constructor
```

Fig. L 11.3 | `EventsFrame.java`. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Events

```

   // ActionListener event handlers
   /* Implement the ActionListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // ItemListener event handlers
   /* Implement the ItemListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // MouseListener event handlers
   /* Implement the MouseListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // MouseMotionListener event handlers
   /* Implement the MouseMotionListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // KeyListener event handlers
   /* Implement the KeyListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // ListSelectionListener event handlers
   /* Implement the ListSelectionListener interface. Display the string representation
      of each event that occurs in the output JTextArea */

   // display event occurred to output
   public void display( String eventName, Object event )
   {
      outputJTextArea.append( String.format( "%s occurred\n%S\n\n",
         eventName, event.toString() ) );
   } // end method display
} // end class EventsFrame
```

Fig. L 11.3 | `EventsFrame.java`. (Part 3 of 3.)

```
// Exercise 11.16 Solution: Events.java
// Program displays events that occur during execution.
import javax.swing.JFrame;

public class Events
{
   public static void main( String args[] )
   {
      EventsFrame eventsFrame = new EventsFrame(); // create EventsFrame
      eventsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      eventsFrame.setSize( 375, 325 ); // set frame size
      eventsFrame.setVisible( true ); // display frame
   } // end main
} // end class Events
```

Fig. L 11.4 | `Events.java`.

Lab Exercises

Name:

Lab Exercise 2 — Events

Problem-Solving Tips

1. The application itself should listen for all events except the clear button's event. Register each listener with `this` as the listener.
2. Every method of an interface must be defined in a class that implements that interface or else a compilation error will occur. So, ensure that you define all the methods specified by the interfaces implemented in this application.
3. In each event-handling method, you should append a string containing information about the event to the output `JTextArea`.
4. Use method `append` from class `JTextArea` to display all the event information. Place newlines between each event string to make the output easier to read.
5. If you have any questions as you proceed, ask your lab instructor for assistance.

Lab Exercises

Name:

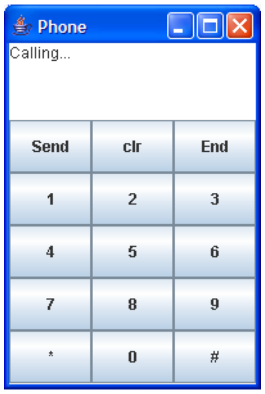
Debugging

Name: ______________________ **Date:** ______________________

Section: ______________________

The program in this section does not compile. Fix all the syntax errors so that the program will compile successfully. Once the program compiles, execute the program, and compare its output with the sample output; then eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code is corrected. The source code is available at `www.deitel.com` and at `www.prenhall.com/deitel`.

Sample Output



Broken Code

```java
// Debugging problem Chapter 11: Phone.java
// Program creates a GUI that resembles a phone with functionality.
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class Phone extends JFrame
{
   private Jbutton keyJButton[];
   private JPanel keyJPanel;
   private JPanel lcdJPanel;
   private JTextArea lcdJTextArea;
   private String lcdOutput = "";
   private int count;

```

Fig. L 11.5 | `Phone.java`. (Part 1 of 3.)

Lab Exercises

Name:

Debugging

```
   // constructor sets up GUI
   public Phone()
   {
      super( "Phone" );

      lcdJTextArea = new JTextArea( 4, 15 );
      lcdJTextArea.setEditable( false );
      lcdJPanel.add( lcdJTextArea );

      keyJButton = new Jbutton[ 15 ];

      // initialize all digit key Buttons
      for ( int i = 3; i <= 11; i++ )
         keyJButton[ i ] = new Jbutton( String.valueOf( i - 2 ) );

      // initialize all non-digit key Buttons
      keyJButton[ 0 ] = new Jbutton( "Send" );
      keyJButton[ 1 ] = new Jbutton( "clr" );
      keyJButton[ 2 ] = new Jbutton( "End" );
      keyJButton[ 12 ] = new Jbutton( "*" );
      keyJButton[ 13 ] = new Jbutton( "0" );
      keyJButton[ 14 ] = new Jbutton( "#" );

      keyJButton[ 0 ].addActionListener(

            public void actionPerformed( ActionEvent e )
            {
               lcdOutput = "Calling...\n\n" + lcdOutput;
               lcdJTextArea.setText( lcdOutput );
            } // end method actionPerformed
         } // end new ActionListener
      ) // end addActionListener call

      keyJButton[ 1 ].addActionListener(

         new ActionListener()
         {
            public void actionPerformed( ActionEvent e )
            {
               if ( lcdOutput.length() == 0 ||
                  lcdOutput.substring( 0, 1 ).equals( "C" ) )
                  return;
               else
               {
                  lcdOutput = lcdOutput.substring( 0, ( lcdOutput.length() - 1 ) );
                  lcdJTextArea.setText( lcdOutput );
               } // end else
            } // end method actionPerformed
         } // end object ActionLstener
      ); // end addActionListener call

```

Fig. L 11.5 | `Phone.java`. (Part 2 of 3.)

Lab Exercises Name:

Debugging

```
      keyJButton[ 2 ].addActionListener(

         new ActionListener()
         {
            public void actionPerformed( ActionEvent e )
            {
               lcdJTextArea.setText( " " );
               lcdOutput = "";
            } // end method actionPerformed
         } // end new ActionListener
      ); // end ActionListener call

      for ( int i = 3; i <= 14; i++ )
      {
         keyJButton[ i ].addActionListener(

            new ActionListener()
            {
               public void actionPerformed( ActionEvent e )
               {
                  lcdOutput += e.getActionCommand();

                  if ( lcdOutput.substring( 0, 1 ).equals( "C" ) )
                      return;

                  lcdJTextArea.append( e.getActionCommand() );
               } // end method actionPerformed
            } // end new ActionListener
         ); // end addActionListener call
      } // end for loop

      // set keyJPanel layout to grid layout
      keyJPanel = new JPanel();
      keyJPanel.setLayout( new GridLayout( 5, 3 ) );

      // add buttons to keyJPanel
      for ( int i = 0; i <= 14; i++ )
         keyJPanel.add( keyJButton[ i ] );

      // add components to container
      add( lcdOutput, BorderLayout.NORTH );
   } // end Phone constructor
} // end class Phone
```

Fig. L 11.5 | `Phone.java`. (Part 3 of 3.)

```
// Debugging problem Chapter 11: PhoneTest.java
// Program creates a GUI that resembles a phone with functionality.
import javax.swing.JFrame;

public class PhoneTest
{
   // execute application
   public static void main( String args[] )
   {
      Phone application = new Phone();
```

Fig. L 11.6 | `PhoneTest.java`. (Part 1 of 2.)

Lab Exercises Name:

Debugging

```
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.setSize( 200, 300 );
        application.setVisible( true );
    } // end main
} // end class PhoneTest
```

Fig. L 11.6 | `PhoneTest.java`. (Part 2 of 2.)

Postlab Activities

Coding Exercises

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

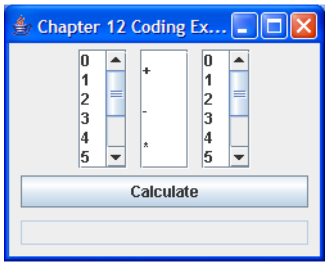
These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have successfully completed the *Prelab Activities* and *Lab Exercises*.

The following application tests class `CodingExercise` *that you will create and enhance in Coding Exercises 1–4.*

```java
// CodingExerciseTest.java
import javax.swing.JFrame;

public class CodingExerciseTest
{
   public static void main( String args[] )
   {
      CodingExercise window = new CodingExercise();
      window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      window.setSize( 250, 200 );
      window.setVisible( true );
   } // end main
} // end class CodingExerciseTest
```

1. Create the following GUI (you will provide functionality later): The GUI consists of three `JList`s: two that contain the numbers 0–9, and one that contains three operations (+, - and *). The GUI should also contain a `JButton` with the label `"Calculate"` and a `JTextField`. Each `JList` should also be contained in a `JScrollPane`. The window shown is separated into two `JPanel`s — the top one contains the three `JScrollPane`s in a `GridLayout`, and the bottom one contains the `"Calculate"` button and the `JTextField` in a `BorderLayout`. You may space and size all the components as you like.



Postlab Activities

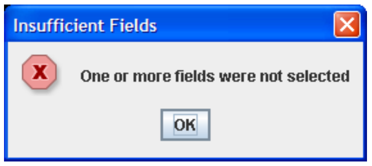
Name:

Coding Exercises

2. Modify the program from *Coding Exercise 1* such that each `JList` is set to `SINGLE_SELECTION`.

3. Modify the program from *Coding Exercise 2* by adding an `ActionListener` to the `"Calculate"` button. When the button is pressed, it should retrieve the values from the three lists and display the calculated value in the `JTextField` (e.g., if `"1"`, `"+"` and `"2"` are selected in the lists, then the `JTextField` should display `"1 + 2 = 3"`).

4. Modify the program from *Coding Exercise 3* so that, when the user clicks the `"Calculate"` button, the program ensures that the user selected a value from each list. If not, the program should display a message telling the user that an item must be selected from each list.


Insufficient Fields

One or more fields were not selected

Postlab Activities

Name:

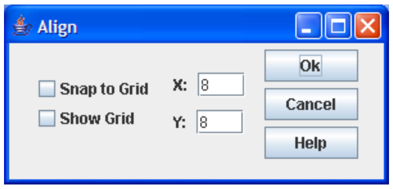
Programming Challenges

Name: ______________________ **Date:** ______________________

Section: ______________________

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a Java program for each of the problems in this section. The answers to these problems are available at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints or sample outputs are provided for each problem to aid you in your programming.

1. Create the following GUI. You do not have to provide any functionality.



[ ] Snap to Grid X: 8
[ ] Show Grid Y: 8

Hints:

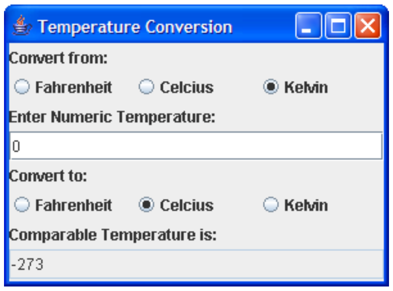
- Your application should use `JPanel`s to arrange the GUI components.
- You will need to use multiple layout managers to properly set up the `JPanel`s.

2. Enhance the temperature conversion application of Exercise 11.12 by adding the Kelvin temperature scale. The application should also allow the user to make conversions between any two scales. Use the following formula for the conversion between Kelvin and Celsius (in addition to the formula in Exercise 11.12):

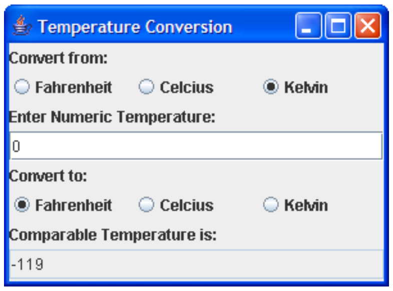
```
Kelvin = Celsius + 273.15
```

Hints:

- Your application should use `JPanel`s to arrange the GUI components.
- First set up the layout with no functionality (i.e., just the look and feel of the application.)
- Next add functionality to the application: Add listeners to all the tool components and a mouse listener for the window.
- To convert from Fahrenheit to Kelvin, first convert from Fahrenheit to Celsius, then convert from Celsius to Kelvin.



Convert from:
Fahrenheit Celcius [✓ Kelvin]
Enter Numeric Temperature:
0
Convert to:
Fahrenheit [✓ Celcius] Kelvin
Comparable Temperature is:
-273



Convert from:
Fahrenheit Celcius [✓ Kelvin]
Enter Numeric Temperature:
0
Convert to:
[✓ Fahrenheit] Celcius Kelvin
Comparable Temperature is:
-119