Chapter 16 - GUI

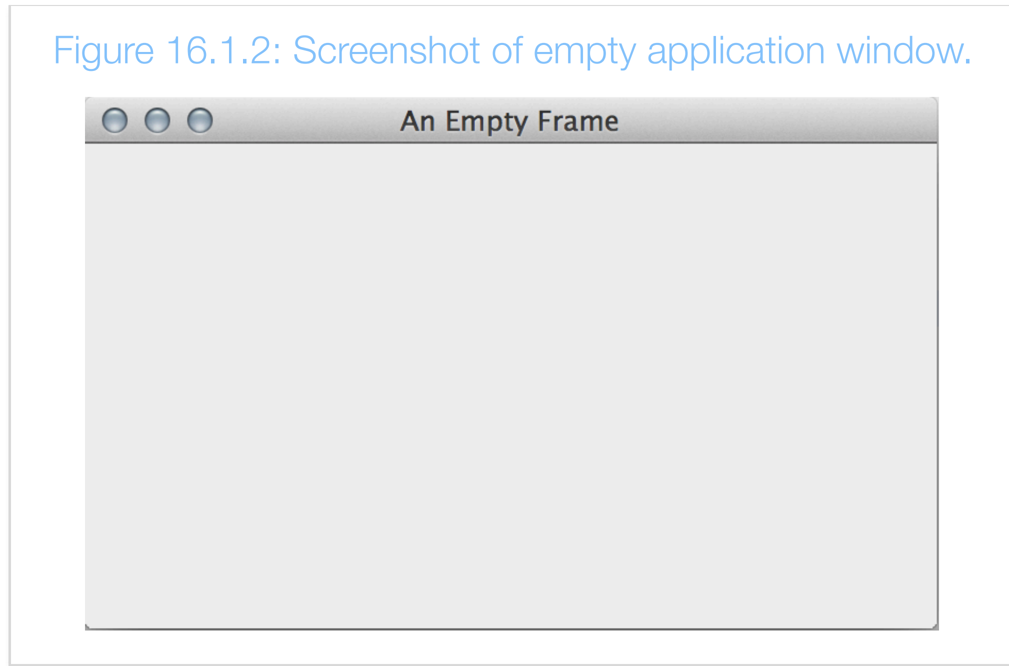
Section 16.1 - Basic graphics

Java supports a set of objects for developing graphical applications. A ***graphical application*** is a program that displays drawings and other graphical objects. Graphical applications display their contents inside a window called a ***frame*** using a ***JFrame*** object. The following program shows how to create and configure an JFrame object to display an empty application window.

Figure 16.1.1: Creating a JFrame object for a graphical application.

```
import javax.swing.JFrame;

public class EmptyFrame {
   public static void main(String[] args) {

      // Construct the JFrame object
      JFrame appFrame = new JFrame();

      // Set the frame's width (400) and height (250) in pixels
      appFrame.setSize(400, 250);

      // Set the frame's title
      appFrame.setTitle("An Empty Frame");

      // Set the program to exit when the user
      // closes the frame
      appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // Make the frame visible to the user
      appFrame.setVisible(true);

      return;
   }
}
```

Figure 16.1.2: Screenshot of empty application window.



Constructing a JFrame object does not immediately display a frame. The program uses the methods supported by the JFrame object to configure and display the frame as follows:

1. **Set the frame's size** by calling the setSize() method with arguments for the width and height, as in `appFrame.setSize(400, 250)`. Forgetting to set the frame's size results in a frame too small to see.
2. **Set the frame's title** by calling the setTitle() method with a String as the argument. Alternatively, the frame's title can be provided as an argument to JFrame's constructor as in `JFrame appFrame = new JFrame("An Empty Frame")`.
3. **Set the frame's closing operation** by calling the setDefaultCloseOperation() method, as in `appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`. This statement configures the program to exit when the user closes the frame.
4. **Make the frame visible** to the user by calling the setVisible() method with a boolean argument of true.

P Participation Activity

16.1.1: Configuring a JFrame.

Select the code statement that would resolve the described problem. Assume an empty JFrame object named appFrame.

#	Question	Your answer
1	The frame window lacks a title. User would like the title to be "My program".	`appFrame.setTitle(My program);`
		`appFrame.setTitle("My program");`
2	The program called the setVisible() method correctly (i.e., `appFrame.setVisible(true);`), but the frame is not visible on the screen. The frame should be 500 pixels wide and 300 pixels tall.	`appFrame.setSize(500, 300);`
		`appFrame.setVisible(false);`
		`appFrame.setSize(300, 500);`
3	The program does not exit when the user closes the frame.	`appFrame.setDefaultCloseOperation();`
		`appFrame.setDefaultCloseOperation(JFr`

A JFrame can be used to draw graphical objects, such as rectangles, circles, and lines. To display graphical objects, a programmer can add a *custom* JComponent object to a frame. A ***JComponent*** is a blank graphical component that a programmer extends (or customizes) with custom code in order to draw basic shapes.

The following program demonstrates how to create a custom class that extends JComponent to draw 2D graphics. Creating a class that extends JComponent involves advanced topics, including defining a class and inheritance, which are discussed elsewhere. For now, the following class can be used as a template.

Figure 16.1.3: Basic example showing how to create a class extending JComponent to draw 2D graphics.

```
import java.awt.Graphics;
import javax.swing.JComponent;

public class MyCustomJComponent extends JComponent {
   public void paintComponent(Graphics g) {
      // Cast to Graphics2D
      Graphics2D graphicsObj = (Graphics2D)g;

      // Write your drawing instructions
   }
}
```

The above code defines a class named MyCustomJComponent that extends JComponent. The class should be saved to a separate file named with the same name, MyCustomJComponent.java. A programmer completes the template by providing custom drawing instructions in the paintComponent() method. For example, the following program extends a JComponent to draw a simple histogram using Rectangle and Color objects.

Figure 16.1.4: Drawing a histogram in a frame.

HistogramComponent.java

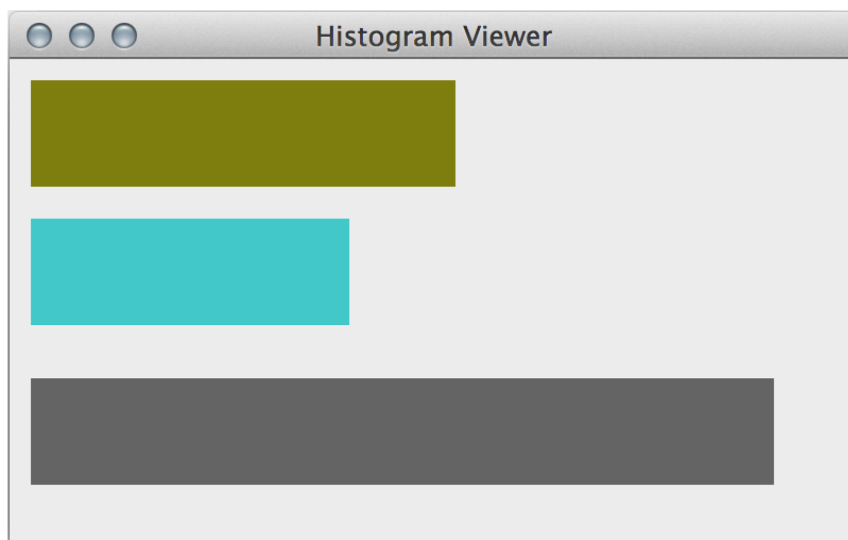
```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

// HistogramComponent extends the functionality of a JComponent
// in order to draw a histogram.
public class HistogramComponent extends JComponent {

   // Paints a histogram with three bins
   public void paintComponent(Graphics g) {
      // Cast to Graphics2D
      Graphics2D graphicsObj = (Graphics2D) g;

      // Draw 1st bin as an olive colored rectangle at (10,10)
      // with width = 200 and height = 50
      Rectangle binRectangle1 = new Rectangle(10, 10, 200, 50);
      Color binColor1 = new Color(128, 128, 0);
      graphicsObj.setColor(binColor1);
      graphicsObj.fill(binRectangle1);

      // Draw 2nd bin as a teal blue rectangle at (10,75)
      // with width = 150 and height = 50
      Rectangle binRectangle2 = new Rectangle(10, 75, 150, 50);
      Color binColor2 = new Color(0, 200, 200);
      graphicsObj.setColor(binColor2);
      graphicsObj.fill(binRectangle2);
```

```
        // Draw 3rd bin as a gray rectangle at (10,140)
        // with width = 350 and height = 50
        Rectangle binRectangle3 = new Rectangle(10, 140, 350, 50);
        Color binColor3 = new Color(100, 100, 100);
        graphicsObj.setColor(binColor3);
        graphicsObj.fill(binRectangle3);

        return;
    }
}
```

HistogramViewer.java

```
import javax.swing.JFrame;

public class HistogramViewer {
    public static void main(String[] args) {
        JFrame appFrame = new JFrame();
        HistogramComponent histogramComponent = new HistogramComponent();

        appFrame.setSize(400, 250);
        appFrame.setTitle("Histogram Viewer");
        appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Add the HistogramComponent object to the frame
        appFrame.add(histogramComponent);

        // Set the frame and its contents visible
        appFrame.setVisible(true);

        return;
    }
}
```

Figure 16.1.5: Screenshot of HistogramViewer application.



The program first creates a HistogramComponent object named histogramComponent and adds the object to the JFrame object using the add() method. Once added, the JFrame automatically calls the histogramComponent objects paintComponent() method whenever the JFrame object is updated, such as when the frame is resized.

The HistogramComponent's paintComponent() uses Rectangle and Color objects to draw a simple histogram with three bins, using the operations:

1. **Cast the Graphics object to Graphics2D**: The statement `Graphics2D graphicsObj = (Graphics2D) g;` converts the original Graphics object argument to a graphics object that supports drawing two-dimensional objects.
2. **Create a Rectangle object**: A ***Rectangle*** object stores the location and size of a rectangle shape. A Rectangle's constructor accepts arguments for location and size (in pixel units) as specified by the constructor definition: `Rectangle(int x, int y, int width, int height)`.
3. **Create a Color object**: A ***Color*** object represents a color in the red, green, blue color space. A Color constructor accepts an integer value between 0 to 255 for each color channel as specified by the constructor definition: `Color(int red, int green, int blue)`. For example, the statement `Color binColor1 = new Color(128, 128, 0);` creates a Color object with an olive color.
4. **Set the color used by the Graphics2D object**: Graphic2D's setColor() method sets the color that the Graphics2D object will use for subsequent drawing operations.
5. **Draw the shape**: A Graphic2D object provides different methods for drawing shapes. The draw() method will draw an outline of a shape, such as a Rectangle object, using the Graphic2D object's current color. The fill() method will draw a shape filling the interior of the shape with the Graphic2D object's current color.

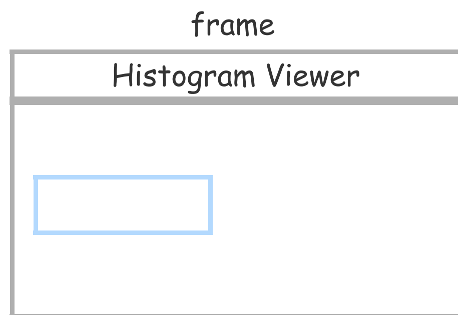
P Participation Activity

16.1.2: Drawing a filled rectangle.

Start

```
Rectangle binRect = new Rectangle(10, 75, 150, 50);
Color binColor = new Color(0, 200, 200);
graphicsObj.setColor(binColor);
graphicsObj.fill(binRect);
```

frame



graphicsObj object

Internal data:
(0, 200, 200)

binRect object

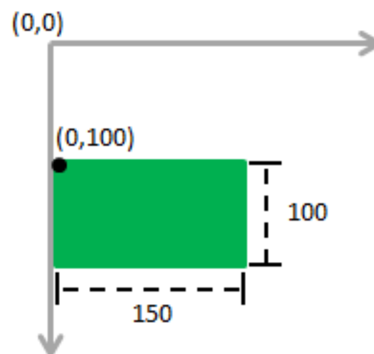
Internal data:
(10, 75)
50
150

binColor object

Internal data:
(0, 200, 200)

The programmer needs to know the positioning coordinate system in order to draw shapes in the intended location. As the following figure illustrates, the top-left corner of a JComponent corresponds to coordinates (0, 0). The x-coordinate increases horizontally to the right and the y-coordinate increases vertically downward.

Figure 16.1.6: Graphics coordinate system.



P Participation Activity

16.1.3: Drawing colored rectangles.

Which code segment (write the number) performs the described operation? Assume each code segment is written within the paintComponent() method of an extended JComponent class, and that the Graphics2D object is called graphicsObj.

A.
```
Rectangle rectangle = new Rectangle(0, 0, 150, 100);
Color color = new Color(0, 255, 0);
graphicsObj.setColor(color);
graphicsObj.fill(rectangle);
```

B.
```
Rectangle rectangle = new Rectangle(0, 100, 200, 200);
Color color = new Color(255, 0, 0);
graphicsObj.setColor(color);
graphicsObj.fill(rectangle);
```

C.
```
Rectangle rectangle = new Rectangle(0, 100, 50, 150);
Color color = new Color(255, 0, 255);
graphicsObj.setColor(color);
graphicsObj.draw(rectangle);
```

#	Question	Your answer
1	Draws a filled in red square.	
2	Draws the outline of a purple rectangle 50 pixels wide and 150 pixels in height.	
3	Draws a rectangle whose top-left corner is located at the origin of the coordinate system.	
4	Draws a green, filled in rectangle.	

A Graphics2D object can draw a variety of shapes, of which some common shapes are summarized below:

Table 16.1.1: Summary of common shapes for drawing.

Shape	Description	Documentation
Rectangle	The Rectangle class for drawing a rectangle.	Oracle's documentation for Rectangle class
RoundRectangle2D	The RoundRectangle2D class for drawing a rectangle with rounded corners.	Oracle's documentation for RoundRectangle2D class
Ellipse2D.Double	The Ellipse2D.Double class for drawing an ellipse with a size and location.	Oracle's documentation for Ellipse2D.Double class
Line2D.Double	The Line2D.Double class for drawing a line between two coordinate points.	Oracle's documentation for Line2D.Double class
Polygon	The Polygon class for drawing a generic polygon with user-specified boundary points.	Oracle's documentation for Polygon class

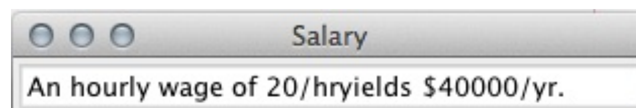
Section 16.2 - Introduction to graphical user interfaces

Java supports a set of components, called ***Swing GUI components***, for the development of custom GUIs. A GUI, or ***graphical user interface***, enables the user to interface with a program via the use of graphical components such as windows, buttons, text boxes, etc. as opposed to text-based interfaces like the traditional command line. The following example calculates a yearly salary based on an hourly wage and utilizes Swing GUI components in order to create a GUI that displays the program's output.

Figure 16.2.1: Displaying a yearly salary using a GUI.

```
import javax.swing.JFrame;
import javax.swing.JTextField;

public class SalaryGUI {
   public static void main(String[] args) {
      int hourlyWage = 0;
      JFrame topFrame = null;          // Application window
      JTextField outputField = null; // Displays output salary

      hourlyWage = 20;

      // Create text field
      outputField = new JTextField();
      // Display program output using the text field
      outputField.setText("An hourly wage of " + hourlyWage + "/hr" +
                          " yields $" + (hourlyWage * 40 * 50) + "/yr.");

      // Prevent user from editing output text
      outputField.setEditable(false);

      // Create window
      topFrame = new JFrame("Salary");

      // Add text field to window
      topFrame.add(outputField);

      // Resize window to fit components
      topFrame.pack();

      // Set program to terminate when window is closed
      topFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // Display window
      topFrame.setVisible(true);

      return;
   }
}
```

Screenshot:



The above program utilizes two basic Swing GUI components: ***JTextField*** and ***JFrame***. The resulting GUI consists of a window (i.e., a JFrame) and a text field (i.e., a JTextField), as illustrated by the screenshot above.

A JTextField is a Swing GUI component that enables a programmer to display a line of text and is available via the import statement `import javax.swing.JTextField;`. The statement `outputField = new JTextField();` creates a JTextField object, which is represented by the

variable outputField. A programmer can then use JTextField's setText() method to specify the text that will be displayed, as in the statement `outputField.setText("An hourly ... ");`. By default, a JTextField allows users to modify the displayed text at runtime for the purposes of input (discussed elsewhere). However, the above program invokes JTextField's setEditable() method with the boolean literal false, as in `outputField.setEditable(false);`, to prevent users from editing the displayed text.

A JFrame is a ***top-level container*** of GUI components and serves as the application's main window. The JFrame class is available to programmers via the import statement `import javax.swing.JFrame;`. The statement `frame = new JFrame("Salary");` creates a window frame titled "Salary", as specified by the String literal within parentheses. A frame must contain all GUI components that should be visible to the user. A programmer uses JFrame's add() method to add GUI components to the frame. For example, the statement `frame.add(outputField);` adds the JTextField component, outputField, to the frame. The outputField text field is contained within the frame and displayed within the application's window.

After adding all GUI components to a frame, a programmer then invokes JFrame's pack() method, as in `frame.pack();`, to automatically resize the frame to fit all of the contained components. Importantly, the pack() method resizes the window according to the current state of the contained components. Thus, modifying, adding, or removing GUI components after the call to pack() may result in a window that is not sized appropriately.

Try 16.2.1: Experimenting with JFrame's pack() method.

JFrame's pack() method uses the preferred size of its contained components in order to determine the appropriate size for the window. Try removing the statement frame.pack() from the above program and observe the effect. Notice how the window no longer displays the entire text of the JTextField component. Instead, the window defaults to a default size without considering the size of the frame's contained components.

Now restore the program to the original state and try moving the statement `outputField.setText("An hourly wage ...");` after the call to pack() (i.e., after the statement frame.pack();). Run the program once again and observe the output. Although the program invoked the pack() method, the text field is not displayed properly within the window. The statement order matters. The pack() method resizes the window according to the current state of the frame's components. Thus, changing the amount of text displayed by a JTextField component after the call to pack() will not automatically resize the window in order to fit the text.

By default, closing a GUI window does not terminate the program. Thus, the statement `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` is required so that the program terminates properly when the GUI window is closed. Lastly, the statement

`frame.setVisible(true);` makes the frame visible on the screen.

In summary, the statements in the program's main() method construct a GUI as outlined by the following procedure:

1. Create GUI components (e.g., JTextField)
2. Create a top-level GUI component container (e.g., JFrame)
3. Add GUI components to the top-level container
4. Configure the top-level container (e.g., set the default close operation)
5. Display the top-level container (e.g., make the frame visible)

P Participation Activity

16.2.1: Using Swing GUI components.

#	Question	Your answer
1	Write a statement that sets the text of a JTextField component named nameField to "Mary".	
2	Given the JFrame variable named frame, write a statement that creates a JFrame with the title "Employees".	
3	Given a JFrame variable named frame and a JTextField variable named nameField, write a statement that adds nameField to the frame.	
4	Given a JFrame variable named frame, write a statement that makes the frame visible.	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials

---

Section 16.3 - Positioning GUI components using a GridBagLayout

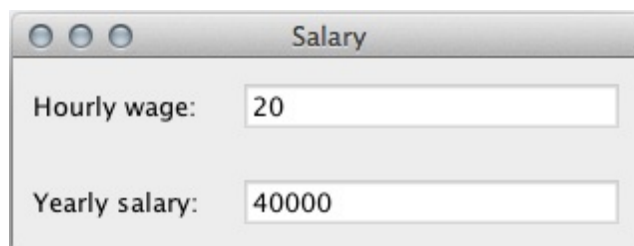
A ***layout manager*** affords programmers control over the positioning and layout of GUI components within a JFrame or other such containers. A ***GridBagLayout*** positions GUI components in a two-dimensional grid and is one of the layout managers supported by Java. The following example demonstrates the use of a GridBagLayout to position GUI components for a program that displays an hourly wage and the associated yearly salary.

Figure 16.3.1: Using a GridBagLayout to arrange GUI components.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SalaryLabelGUI {

   public static void main(String[] args) {
      int hourlyWage = 0;
      JFrame topFrame = null;                    // Application window
      JLabel wageLabel = null;                   // Label for hourly salary
      JLabel salLabel = null;                    // Label for yearly salary
      JTextField salField = null;                // Displays hourly salary
      JTextField wageField = null;               // Displays yearly salary
      GridBagConstraints layoutConst = null; // GUI component layout

      hourlyWage = 20;

      // Set hourly and yearly salary
      wageLabel = new JLabel("Hourly wage:");
      salLabel = new JLabel("Yearly salary:");

      wageField = new JTextField(15);
      wageField.setEditable(false);
      wageField.setText(Integer.toString(hourlyWage));

      salField = new JTextField(15);
      salField.setEditable(false);
      salField.setText(Integer.toString((hourlyWage * 40 * 50)));

      // Create frame and add components using GridBagLayout
      topFrame = new JFrame("Salary");

      // Use a GridBagLayout
      topFrame.setLayout(new GridBagLayout());
```

```
        // Create GridBagConstraints
        layoutConst = new GridBagConstraints();

        // Specify component's grid location
        layoutConst.gridx = 0;
        layoutConst.gridy = 0;

        // 10 pixels of padding around component
        layoutConst.insets = new Insets(10, 10, 10, 10);

        // Add component using the specified constraints
        topFrame.add(wageLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.gridx = 1;
        layoutConst.gridy = 0;
        layoutConst.insets = new Insets(10, 10, 10, 10);
        topFrame.add(wageField, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.gridx = 0;
        layoutConst.gridy = 1;
        layoutConst.insets = new Insets(10, 10, 10, 10);
        topFrame.add(salLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.gridx = 1;
        layoutConst.gridy = 1;
        layoutConst.insets = new Insets(10, 10, 10, 10);
        topFrame.add(salField, layoutConst);

        // Terminate program when window closes
        topFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Resize window to fit components
        topFrame.pack();

        // Display window
        topFrame.setVisible(true);

        return;
    }
}
```

Screenshot:



The above program displays an hourly wage and a yearly salary using the JTextField component's wageField and salaryField respectively. The statements creating the JTextField components (e.g.,

`wageField = new JTextField(15);`) now specify the fields' widths in number of columns, where a column is proportional to a character's pixel width given a particular font.

Additionally, the program contains two JLabel Swing GUI components, which allow programmers to display text that is typically used for the purposes of describing, or labeling, other GUI components. For example, the statement `wageLabel = new JLabel("Hourly wage:");` creates a JLabel component that describes the value displayed in the wageField. The JLabel class is available to programmers via the import statement `import javax.swing.JLabel;`.

Because the above program uses more than one Swing GUI component, a layout manager is necessary in order to specify the relative position of each component within the frame. A GridBagLayout is a layout manager that allows programmers to place components in individual cells within a two-dimensional grid. Each cell of this grid is indexed using one number for the column, x, and another number for the row, y. The top-left cell is at location (x=0, y=0), and column numbers increase going right, while row numbers increase going down. The programmer is additionally able to add padding (i.e., empty space) between Swing GUI components in order to make the GUI easier to understand as well as more aesthetically pleasing. The following animation illustrates the process of specifying the layout for each GUI component.

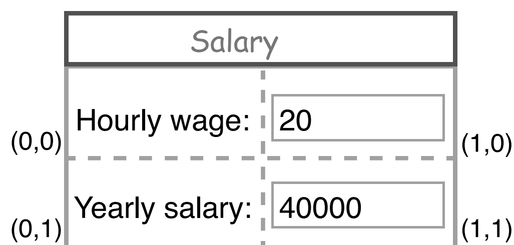
Participation Activity

16.3.1: Specifying layouts for GUI components.

Start

```
wageLabel = new JLabel("Hourly wage:");
salLabel= new JLabel("Yearly salary:");


wageField = new JTextField(15);
wageField.setEditable(false);
wageField.setText(Integer.toString(wage));

salField = new JTextField(15);
salField.setEditable(false);
salField.setText(Integer.toString((wage * 40 * 50)));

frame = new JFrame("Salary");
frame.setLayout(new GridBagLayout());
layoutConst = new GridBagConstraints();

layoutConst.gridx = 0;
layoutConst.gridy = 0;
layoutConst.insets = new Insets(10,10,10,10);
frame.add(wageLabel,layoutConst);

layoutConst.gridx = 1;
layoutConst.gridy = 0;
layoutConst.insets = new Insets(10,10,10,10);
frame.add(wageField,layoutConst);

// Add salary label and field
```

92	Hourly wage:	JLabel object
93	Yearly salary:	JLabel object
94	20	JTextField object
95	40000	JTextField object



A programmer must assign a layout manager to the JFrame component before adding components and specifying their layout via the add() method. The statement `frame.setLayout(new GridBagLayout());` utilizes JFrame's setLayout() method to set a GridBagLayout as the frame's layout manager. The statement `layoutConst = new GridBagConstraints();` creates a GridBagConstraints variable named layoutConst that a programmer can use to specify layout constraints. Layout constraints include the grid location and padding around a GUI component, among others. Although the program could have created a single layoutConst object for all GUI components, notice that the program creates a separate layoutConst object for each GUI component via the statement `layoutConst = new GridBagConstraints();`. Creating a separate layout constraints object for each individual GUI component is a good practice because it forces the programmer to completely specify the layout for each component. To use the GridBagLayout and GridBagConstraints classes, a program must include the statements `import java.awt.GridBagLayout;` and `import java.awt.GridBagConstraints;`, respectively.

A programmer may place a component in any row or column. The wageLabel component, for example, has a grid coordinate location of (x=0, y=0), i.e., the top-left corner, due to the statements `layoutConst.gridx = 0;` and `layoutConst.gridy = 0;`, which specify the component's x and y coordinates respectively. Additionally, all GUI components in the above program use a padding of 10 pixels in all four cardinal directions (i.e., top, left, bottom, and right). This padding is specified by the statement `layoutConst.insets = new Insets(10, 10, 10, 10);`, where the four numbers within parentheses denote the padding in the top, left, bottom, and right directions, respectively. To use Insets, the program must include the statement `import javax.awt.Insets;`. Note that these layout constraints are only applied after invoking JFrame's add() method with both a component and its corresponding GridBagConstraints variable, as in `frame.add(wageLabel, layoutConst);`.

The following table summarizes common layout constraints. Refer to How to Use GridBagLayout from Oracle's Java tutorials for a more comprehensive description of all available layout constraints.

Table 16.3.1: Common layout constraints specified with a GridBagConstraints object.

Constraint	Description	Sample usage
gridx, gridy	Used to specify the location (i.e., row and column) of a component	`layoutConst.gridx = 10;`
insets	Used to specify the minimum pixel padding in all four cardinal directions between a component and the edge of its containing cell.	`layoutConst.insets = new Insets(topPad, leftPad,`
gridwidth, gridheight	Used to specify the width (or height) of a component in number of cells	`layoutConst.gridwidth = 2;`

The following example demonstrates an alternative method of coding the previous GUI. This program uses an advanced concept known as inheritance, which is discussed in greater detail elsewhere, in order to define a custom JFrame component (i.e., the custom JFrame component is a type of JFrame).

Figure 16.3.2: Using an alternative coding style that defines a custom JFrame class.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SalaryLabelFrame extends JFrame {
   private JLabel wageLabel;     // Label for hourly salary
   private JLabel salLabel;      // Label for yearly salary
   private JTextField salField;  // Displays hourly salary
   private JTextField wageField; // Displays yearly salary

   /* Constructor initializes the SalaryLabelFrame,
      creates GUI components, and adds them
      using a GridBagLayout. */
   SalaryLabelFrame() {
      int hourlyWage = 20;               // Hourly wage
      GridBagConstraints layoutConst;  // Used to specify GUI component layout

      // Set frame's title
      setTitle("Salary");

      // Set hourly and yearly salary
      wageLabel = new JLabel("Hourly wage:");
      salLabel = new JLabel("Yearly salary:");

      wageField = new JTextField(15);
      wageField.setEditable(false);
      wageField.setText(Integer.toString(hourlyWage));

      salField = new JTextField(15);
      salField.setEditable(false);
      salField.setText(Integer.toString((hourlyWage * 40 * 50)));

      // Use a GridBagLayout
      setLayout(new GridBagLayout());

      // Create GridBagConstraints
      layoutConst = new GridBagConstraints();

      // Specify component's grid location
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;

      // 10 pixels of padding around component
      layoutConst.insets = new Insets(10, 10, 10, 10);

      // Add component using the specified constraints
      add(wageLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      layoutConst.insets = new Insets(10, 10, 10, 10);
      add(wageField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
```

```
         layoutConst.insets = new Insets(10, 10, 10, 10);
         add(salLabel, layoutConst);

         layoutConst = new GridBagConstraints();
         layoutConst.gridx = 1;
         layoutConst.gridy = 1;
         layoutConst.insets = new Insets(10, 10, 10, 10);
         add(salField, layoutConst);
      }

      /* Creates a SalarLabelFrame and makes it visible */
      public static void main(String[] args) {
         // Creates SalaryLabelFrame and its components
         SalaryLabelFrame myFrame = new SalaryLabelFrame();

         // Terminate program when window closes
         myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

         // Resize window to fit components
         myFrame.pack();

         // Display window
         myFrame.setVisible(true);

         return;
      }
   }
```

The program specifies that the SalaryLabelFrame class is a special kind of JFrame by appending `extends JFrame` after the class name, i.e., `class SalaryLabelFrame extends JFrame {`. The extends keyword indicates that SalaryLabelFrame inherits the functionality of a JFrame, and thus a programmer can use a SalaryLabelFrame component for the same purposes as a JFrame component, i.e., as a container of GUI components.

Additionally, defining a custom JFrame class enables a programmer to augment (i.e., extend) a normal JFrame with additional functionality. For example, the SalaryLabelFrame class contains a special method called a constructor, highlighted in yellow. The ***constructor*** contains all the code necessary to create and arrange the GUI components (i.e., the labels and text fields) and is automatically called when an instance of the containing class is created. The statement `SalaryLabelFrame myFrame = new SalaryLabelFrame();` automatically calls the SalaryLabelFrame() constructor. Thus, creating a SalaryLabelFrame component also initializes all GUI components.

Because SalaryLabelFrame's constructor handles the initialization and layout of all Swing GUI components, the main() method need only create a SalaryLabelFrame component named myFrame. A SalaryLabelFrame component is created using the statement `SalaryLabelFrame mFrame = new SalaryLabelFrame();`, setting the frame's default close operation, resizing the frame using pack(), and ultimately displaying the window. This alternative GUI coding style has the advantage of creating a more modular and readable program. That is, the code for constructing the GUI and the GUI's components is neatly separated from any higher-level logic

executing within main().

Note that the code within the constructor directly invokes JFrame's methods, e.g., `add(wageLabel, layoutConst);`, without the need to create a separate JFrame component because SalaryLabelFrame is a type of JFrame. The statement `setTitle("Salary");` similarly calls JFrame's setTitle() method directly in order to set the window's title. Lastly, the variables for the GUI components (e.g., wageLabel and salLabel) are declared outside both the constructor and main() for reasons that are discussed elsewhere.

P Participation Activity

16.3.2: Using labels and specifying layouts.

#	Question	Your answer
1	Given the JLabel variable named nameField, write a statement that creates a JLabel with the text "Name:".	
2	Using the GridBagConstraints variable named layoutConstraints, write two statements that would constrain a GUI component's location to the top left corner of the grid.	
3	Using the GridBagConstraints variable named layoutConstraints, write a statement that would add 5 pixels of padding to the left and right of a GUI component. The top and bottom edges of the component should not have padding.	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials
- How to Use GridBagLayout from Oracle's Java tutorials

Section 16.4 - GUI input and ActionListeners

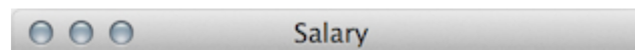
Several Swing GUI components, such as a JTextField, support user input. The following example uses an editable text field to enable GUI users to enter an hourly wage value as an input for the calculation of a yearly salary.

Figure 16.4.1: Using a JTextField to enter a wage for a yearly salary calculation.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SalaryCalcFrame extends JFrame implements ActionListener {
   private JLabel wageLabel;     // Label for hourly salary
   private JLabel salLabel;      // Label for yearly salary
   private JTextField salField;  // Displays hourly salary
   private JTextField wageField; // Displays yearly salary


   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   SalaryCalcFrame() {
      // Used to specify GUI component layout
      GridBagConstraints layoutConst = null;

      // Set frame's title
      setTitle("Salary");

      wageLabel = new JLabel("Hourly wage:");
      salLabel = new JLabel("Yearly salary:");

      // Set hourly and yearly salary
      wageField = new JTextField(15);
      wageField.setEditable(true);
      wageField.setText("0");
      wageField.addActionListener(this);

      salField = new JTextField(15);
      salField.setEditable(false);

      // Use a GridBagLayout
      setLayout(new GridBagLayout());
      layoutConst = new GridBagConstraints();

      // Specify component's grid location
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
```

```
         // 10 pixels of padding around component
         layoutConst.insets = new Insets(10, 10, 10, 10);

         // Add component using the specified constraints
         add(wageLabel, layoutConst);

         layoutConst = new GridBagConstraints();
         layoutConst.gridx = 1;
         layoutConst.gridy = 0;
         layoutConst.insets = new Insets(10, 10, 10, 10);
         add(wageField, layoutConst);

         layoutConst = new GridBagConstraints();
         layoutConst.gridx = 0;
         layoutConst.gridy = 1;
         layoutConst.insets = new Insets(10, 10, 10, 10);
         add(salLabel, layoutConst);

         layoutConst = new GridBagConstraints();
         layoutConst.gridx = 1;
         layoutConst.gridy = 1;
         layoutConst.insets = new Insets(10, 10, 10, 10);
         add(salField, layoutConst);
      }

      /* Method is automatically called when an event
       occurs (e.g, Enter key is pressed) */
      @Override
      public void actionPerformed(ActionEvent event) {
         String userInput = ""; // User specified hourly wage
         int hourlyWage = 0;    // Hourly wage

         // Get user's wage input
         userInput = wageField.getText();

         // Convert from String to an integer
         hourlyWage = Integer.parseInt(userInput);

         // Display calculated salary
         salField.setText(Integer.toString(hourlyWage * 40 * 50));

         return;
      }

      /* Creates a SalaryCalculatorFrame and makes it visible */
      public static void main(String[] args) {
         // Creates SalaryLabelFrame and its components
         SalaryCalcFrame myFrame = new SalaryCalcFrame();

         myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         myFrame.pack();
         myFrame.setVisible(true);

         return;
      }
   }
```

Screenshot:



A programmer can configure a JTextField component to allow users to edit the field's text by calling JTextField's setEditable() method with the argument true, as in `wageField.setEditable(true);`.

GUI components that support user input generate ***action events*** that notify the program that an input has been received and is ready for processing. For example, when the user presses the Enter key while typing input within a text field, the underlying JTextField class generates an action event that informs the program of the new input value (e.g., a new hourlyWage value).

An ***ActionListener*** object is used in a program to handle action events and defines how the program should respond to action events. The above program's SalaryCalcFrame class is both a custom JFrame (i.e., SalaryCalcFrame extends JFrame) and a custom ActionListener (i.e., SalaryCalcFrame implements ActionListener). Note that proper use of the keywords "implements" and "extends" is related to the distinction between a class (e.g., JFrame) and an interface (e.g., ActionListener), both of which are defined and discussed elsewhere. Classes that implement the ActionListener interface must define the actionPerformed() method in order to define how the class should react to an action event.

The SalaryCalcFrame class's actionPerformed() method extracts the String representation of the input hourlyWage using JTextField's getText() method, as in `userInput = wageField.getText();`. This input String is then converted into an integer value via the statement `hourlyWage = Integer.parseInt(userInput);`. Finally, the yearly salary is calculated and displayed by calling salField's setText() method. Use of the ActionListener and ActionEvent classes requires the inclusion of the statements `import java.awt.event.ActionListener;` and `import java.awt.event.ActionEvent;`.

As programs may contain multiple ActionListeners, the programmer must specify the particular ActionListener responsible for handling a GUI component's action events. A programmer can call a GUI component's addActionListener() method in order to register a suitable ActionListener. For example, the statement `wageField.addActionListener(this);` registers the current SalaryCalcFrame object, which is indicated by the `this` keyword, as the ActionListener for wageField. Consequently, the JVM will automatically call the SalaryCalcFrame object's actionPerformed() method in order to calculate a salary when a user provides a hourlyWage value and presses the Enter key within the wageField.

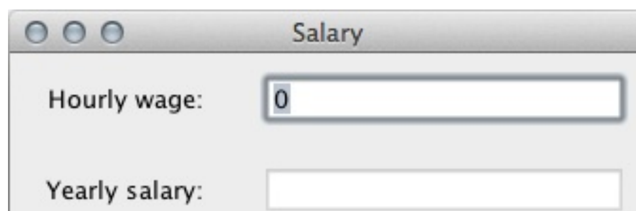
While the above program associates an ActionListener with the wageField in order to detect when the user presses the Enter key, the following program instead uses a button to trigger the yearly salary calculation. Pressing a button within a GUI is often more intuitive than pressing the Enter key.

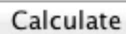
Figure 16.4.2: Using a JButton to trigger a yearly salary calculation.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SalaryCalcButtonFrame extends JFrame implements ActionListener {
   private JLabel wageLabel;     // Label for hourly salary
   private JLabel salLabel;      // Label for yearly salary
   private JTextField salField;  // Displays hourly salary
   private JTextField wageField; // Displays yearly salary
   private JButton calcButton;   // Triggers salary calculation

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   SalaryCalcButtonFrame() {
      // Used to specify GUI component layout
      GridBagConstraints positionConst = null;

      // Set frame's title
      setTitle("Salary");

      // Set hourly and yearly salary labels
      wageLabel = new JLabel("Hourly wage:");
      salLabel = new JLabel("Yearly salary:");

      wageField = new JTextField(15);
      wageField.setEditable(true);
      wageField.setText("0");

      salField = new JTextField(15);
      salField.setEditable(false);

      // Create a "Calculate" button
      calcButton = new JButton("Calculate");

      // Use "this" class to handle button presses
      calcButton.addActionListener(this);

      // Use a GridBagLayout
      setLayout(new GridBagLayout());
      positionConst = new GridBagConstraints();

      // Specify component's grid location
      positionConst.gridx = 0;
      positionConst.gridy = 0;

      // 10 pixels of padding around component
      positionConst.insets = new Insets(10, 10, 10, 10);

      // Add component using the specified constraints
      add(wageLabel, positionConst);

      positionConst.gridx = 1;
```

```
      positionConst.gridx = 1;
      positionConst.gridy = 0;
      positionConst.insets = new Insets(10, 10, 10, 10);
      add(wageField, positionConst);

      positionConst.gridx = 0;
      positionConst.gridy = 1;
      positionConst.insets = new Insets(10, 10, 10, 10);
      add(salLabel, positionConst);

      positionConst.gridx = 1;
      positionConst.gridy = 1;
      positionConst.insets = new Insets(10, 10, 10, 10);
      add(salField, positionConst);

      positionConst.gridx = 0;
      positionConst.gridy = 2;
      positionConst.insets = new Insets(10, 10, 10, 10);
      add(calcButton, positionConst);
   }

   /* Method is automatically called when an event
      occurs (e.g, button is pressed) */
   @Override
   public void actionPerformed(ActionEvent event) {
      String userInput = ""; // User specified hourly wage
      int hourlyWage = 0;    // Hourly wage

      // Get user's wage input
      userInput = wageField.getText();

      // Convert from String to an integer
      hourlyWage = Integer.parseInt(userInput);

      // Display calculated salary
      salField.setText(Integer.toString(hourlyWage * 40 * 50));

      return;
   }

   /* Creates a SalaryCalculatorFrame and makes it visible */
   public static void main(String[] args) {
      // Creates SalaryLabelFrame and its components
      SalaryCalcButtonFrame myFrame = new SalaryCalcButtonFrame();

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.setVisible(true);

      return;
   }
}
```

Screenshot:





The above program utilizes a JButton to allow the user to trigger the calculation and display the yearly salary. A ***JButton*** is a Swing GUI component that represents a labelled button. To use a JButton, the program must include the import statement `import javax.swing.JButton;`.

The program creates a JButton named calcButton using the statement `calcButton = new JButton("Calculate");`, where the String literal between the parentheses is the button's label. Additionally, the program adds the SalaryCalcButtonFrame object as calcButton's ActionListener via the statement `calcButton.addActionListener(this);`. Thus, when the user presses the calcButton, which generates an action event, the actionPerformed() method within the SalaryCalcButtonFrame class automatically receives the action event and displays the calculated salary.

Notice that the program declares the variables for the GUI components (e.g., wageLabel, salLabel) outside the methods and constructor, but still within the class definition. These variables are known as fields. Any method or constructor defined within the class has access to these fields. The program defines all GUI components as fields so that both the constructor and the actionPerformed() method can access them. Classes and fields are discussed in other sections.

Participation Activity

16.4.1: User input with JTextFields and JButtons.

#	Question	Your answer
1	Given the JTextField variable named weeklyHoursField, write a statement that makes weeklyHoursField editable by users.	
2	Given the JTextField variable named weeklyHoursField, write a statement that adds an ActionListener to weeklyHoursField. Use the "this" keyword.	
3	Write a statement that gets the input text from weeklyHoursField and stores it in a variable called userInput.	
4	Using a JButton variable named convertButton, write a statement that creates a JButton with the label "Convert!".	
5	Given the JButton variable named convertButton, write a statement that adds an ActionListener to convertButton. Use the "this" keyword.	

Section 16.5 - GUI input with formatted text fields

A ***JFormattedTextField*** is a Swing GUI component that extends a JTextField in order to enable a programmer to specify the appropriate types and sequence of characters (i.e., the character format) that a text field component can display or accept as input.

The following example reports the time required to travel a user-specified distance (in miles) based on

the mode of transportation. The program uses a JFormattedTextField component to ensure the user enters a valid input that represents a number.

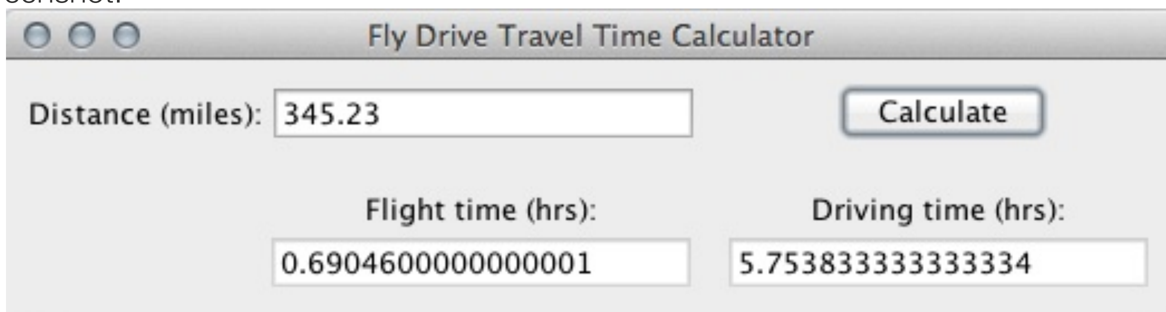
Figure 16.5.1: Using a JFormattedTextField to enter a formatted distance value for a travel time calculation.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class FlyDriveFrame extends JFrame implements ActionListener {
   private JButton calcButton;             // Triggers time calculation
   private JLabel distLabel;               // Label for distance input
   private JLabel hrsFlyLabel;             // Label for fly time
   private JLabel hrsDriveLabel;           // Label for drive time
   private JTextField hrsFlyField;         // Displays fly time
   private JTextField hrsDriveField;       // Displays drive time
   private JFormattedTextField distField; // Holds distance input

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   FlyDriveFrame() {
      // Used to specify GUI component layout
      GridBagConstraints layoutConst = null;

      // Set frame's title
      setTitle("Fly Drive Travel Time Calculator");

      // Create labels
      distLabel = new JLabel("Distance (miles):");
      hrsFlyLabel = new JLabel("Flight time (hrs):");
      hrsDriveLabel = new JLabel("Driving time (hrs):");

      // Create button and add action listener
      calcButton = new JButton("Calculate");
      calcButton.addActionListener(this);

      // Create flight time filed
      hrsFlyField = new JTextField(15);
      hrsFlyField.setEditable(false);

      // Create driving time field
      hrsDriveField = new JTextField(15);
      hrsDriveField.setEditable(false);

      // Create and set-up an input field for numbers (not text)
      distField = new JFormattedTextField(NumberFormat.getNumberInstance());
      distField.setEditable(true);
      distField.setText("0");
```

```
      distField.setText("0");
      distField.setColumns(15); // Initial width of 10 units

      // Use a GridBagLayout
      setLayout(new GridBagLayout());

      // Specify component's grid location
      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 1);
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
      add(distLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 10, 10);
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      add(distField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 5, 10, 10);
      layoutConst.gridx = 2;
      layoutConst.gridy = 0;
      add(calcButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 0, 1, 10);
      layoutConst.gridx = 1;
      layoutConst.gridy = 1;
      add(hrsFlyLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 0, 10, 10);
      layoutConst.gridx = 1;
      layoutConst.gridy = 2;
      add(hrsFlyField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 0, 1, 10);
      layoutConst.gridx = 2;
      layoutConst.gridy = 1;
      add(hrsDriveLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 0, 10, 10);
      layoutConst.gridx = 2;
      layoutConst.gridy = 2;
      add(hrsDriveField, layoutConst);
   }

   /* Method is automatically called when an event
      occurs (e.g, Enter key is pressed) */
   @Override
   public void actionPerformed(ActionEvent event) {
      double totMiles = 0; // Distance to travel
      double hrsFly = 0;   // Corresponding hours to fly
      double hrsDrive = 0; // Corresponding hours to drive

      // Get value from distance field
      totMiles = ((Number) distField.getValue()).doubleValue();

      // Check if miles input is positive
```

```
      // Check if miles input is positive
      if (totMiles >= 0.0) {
         hrsFly = totMiles / 500.0;
         hrsDrive = totMiles / 60.0;

         hrsFlyField.setText(Double.toString(hrsFly));
         hrsDriveField.setText(Double.toString(hrsDrive));
      }
      else {
         // Show failure dialog
         JOptionPane.showMessageDialog(this, "Enter a positive distance value!");
      }

      return;
   }

   /* Creates a FlyDriveFrame and makes it visible */
   public static void main(String[] args) {
      // Creates FlyDriveFrame and its components
      FlyDriveFrame myFrame = new FlyDriveFrame();

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.setVisible(true);

      return;
   }
}
```

Screenshot:



As illustrated by the screenshot above, the GUI consists of an input text field for distance input in miles, a button to trigger the travel time calculations, and two non-editable text fields to display the calculated driving and flying travel times.

The program creates a JFormattedTextField object, assigning its reference to the variable distField, via the statement
`distField = new JFormattedTextField(NumberFormat.getNumberInstance());`. The item within the parentheses should be a Format object. A ***Format object*** specifies the formatting requirements for any string that the JFormattedTextField component displays. The statement `NumberFormat.getNumberInstance()` creates a ***NumberFormat*** object, which specifies the formatting requirements a string must meet in order to represent a real number (e.g., 1, 1.1, -3.14, etc.).

The JFormattedTextField class ensures input validity by keeping track of the most recent, correctly formatted, value typed into (or displayed by) the JFormattedTextField component, and discarding any incorrectly formatted inputs. If the user enters an invalid string such as "two", the JFormattedTextField does not update the stored value because the string "two" does not meet the formatting requirements specified by the aforementioned NumberFormat object. On the other hand, a valid input such as "3.14" causes the JFormattedTextField to update its stored value to "3.14". Thus, when the user presses the calculate button, the program will extract only the most recent valid value from the text field.

Try 16.5.1: Experimenting with partially correct formatted inputs.

Using the above FlyDriveFrame program, enter the input "31 asd 32". Notice that the JFormattedTextField component only extracts the first valid number from this incorrectly formatted input string. Also, notice that the JFormattedTextField component automatically formats the displayed text when the user clicks on another component (e.g., the button).

The program extracts the typed input from the JFormattedTextField component using JFormattedTextField's getValue() method. The program then converts the returned value to a Number and uses the Number class's doubleValue() method to attain the actual distance value as a double, resulting in the statement `miles = ((Number)distField.getValue()).doubleValue();`. This single statement encompasses several topics, such as objects, classes, and method chaining, which are discussed elsewhere.

The getValue() method delegates the task of converting the returned value into the correct type to the programmer so that the method remains compatible with a variety of different built-in Format objects such as NumberFormat, DateFormat, and MessageFormat, or any other custom Format objects defined by the programmer. Note that this section only discusses the NumberFormat, which provides formatting support for real numbers, integers, currency, and percentages. The following table lists and describes such NumberFormat instances. For information regarding the other Format objects, refer to Formatting from Oracle's Java tutorials.

Table 16.5.1: Common NumberFormat instances.

Number format instance	Description	Sample usage
Number	A general purpose number format to represent real numbers such as 1.25	`distField = new JFormattedTextField(NumberForm`
Integer	A number format to represent integers such as 10	`indexField = new JFormattedTextField(NumberFor`
Currency	A number format to represent numbers as currency values such as $1.99	`amountField = new JFormattedTextField(NumberFo`
Percentage	A number format to represent numbers as percentages such as 10%	`interestField = new JFormattedTextField(Number`

Try 16.5.2: Replacing JFormattedTextField with a JTextField.

Modify the above program to use a JTextField component, instead of a JFormattedTextField component, for the distance input. Remember to use the getText() method, as opposed to the getValue() method, in order to extract the distance, as in `Double.parseDouble(distField.getText());`. Compile and run the program. Enter an invalid input value such as "text". You should notice several error messages printed on the command-line, as a regular JTextField does not discard non-numerical values.

Because distField allows all numerical inputs, including negative values, the program contains additional code within the actionPerformed() method that checks for negative distance values. If the user enters a negative distance value, the program skips the travel time calculations and instead displays an error message in a separate window called a message dialog. A ***message dialog*** is a separate a window used to display a simple message. The statement `JOptionPane.showMessageDialog(this, "Enter a positive distance value!");` creates a message dialog window that displays the message specified by the String literal, or variable, within the parentheses. In this case, the message tells the user to "Enter a positive distance value!", as shown in the following figure.

Figure 16.5.2: Message dialog: Example screenshot informing user to enter a positive value.

A message dialog is associated with a parent JFrame. Thus, the statement `JOptionPane.showMessageDialog(this, "Enter a positive distance value!");` specifies the current FlyDriveFrame object in the message dialog's parent frame using the "this" keyword. Closing or terminating a FlyDriveFrame component causes its message dialog to exit as well. Refer to Oracle's Java JOptionPane class specification for information on other types of dialog windows.

P Participation Activity

16.5.1: Using JFormattedTextFields and message dialogs.

#	Question	Your answer
1	Using a JFormattedTextField variable named numItersField, write a statement that creates a JFormattedTextField that uses a NumberFormat to display integers.	
2	Write a single statement that gets the value of a JFormattedTextField called speedField as a double and stores it in a variable of type double called carSpeed.	
3	Write a statement that displays a message dialog with the message "Invalid action!". Use the this operator to denote the dialog's parent frame.	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials
- Formatting from Oracle's Java tutorials
- Oracle's Java JOptionPane class specification

Section 16.6 - GUI input with JSpinners

A ***JSpinner*** is a Swing GUI component that supports user input by enabling the user to select, or enter, a specific value from within a predetermined range of values. A JSpinner supports dual functionalities, allowing the user to enter a value into a formatted text field or cycle through available

values by pressing one of two buttons.

The following demonstrates the use of a JSpinner through an example that converts a dog's age into "human years", e.g., a nine year old dog is approximately 57 years old in human years. The program uses a JSpinner component to enable the user to enter a value for the dog's age, such that the age falls within a range of 0 to 30 years.

Figure 16.6.1: Using a JSpinner to enter a dog's age for a GUI that converts a dog's age into human years.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.SpinnerNumberModel;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class DogYearsFrame extends JFrame implements ChangeListener {
   private JSpinner yearsSpinner;     // Triggers travel time calculation
   private JTextField ageHumanField; // Displays dog's age in human years
   private JLabel yearsLabel;         // Label for dog years
   private JLabel ageHumanLabel;      // Label for human years

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   DogYearsFrame() {
      int initYear = 0; // Spinner initial value display
      int minYear = 0;  // Spinner min value
      int maxYear = 30; // Spinner max value
      int stepVal = 1;  // Spinner step

      // Used to specify GUI component layout
      GridBagConstraints layoutConst = null;

      // Specifies the types of values displayed in spinner
      SpinnerNumberModel spinnerModel = null;

      // Set frame's title
      setTitle("Dog's age in human years");

      // Create labels
      yearsLabel = new JLabel("Select dog's age (years):");
      ageHumanLabel = new JLabel("Age (human years):");

      // Create a spinner model, the spinner, and set the change listener
      spinnerModel = new SpinnerNumberModel(initYear, minYear, maxYear, stepVal);
      yearsSpinner = new JSpinner(spinnerModel);
      yearsSpinner.addChangeListener(this);

      // Create field
      ageHumanField = new JTextField(15);
      ageHumanField.setEditable(false);
```

```
      ageHumanField.setText("0 - 15");

      // Use a GridBagLayout
      setLayout(new GridBagLayout());

      // Specify component's grid location
      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
      add(yearsLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      add(yearsSpinner, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
      add(ageHumanLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 1;
      add(ageHumanField, layoutConst);
   }

   @Override
   public void stateChanged(ChangeEvent event) {
      Integer dogAgeYears = 0; // Dog age input

      dogAgeYears = (Integer) yearsSpinner.getValue();

      // Choose output based on dog's age component
      switch (dogAgeYears) {
         case 0:
            ageHumanField.setText("0 - 15");
            break;

         case 1:
            ageHumanField.setText("15");
            break;

         case 2:
            ageHumanField.setText("24");
            break;

         case 3:
            ageHumanField.setText("28");
            break;

         case 4:
            ageHumanField.setText("32");
            break;
```

```
                  break;

               case 5:
                  ageHumanField.setText("37");
                  break;

               case 6:
                  ageHumanField.setText("42");
                  break;

               case 7:
                  ageHumanField.setText("47");
                  break;

               case 8:
                  ageHumanField.setText("52");
                  break;

               case 9:
                  ageHumanField.setText("57");
                  break;

               case 10:
                  ageHumanField.setText("62");
                  break;

               case 11:
                  ageHumanField.setText("67");
                  break;

               case 12:
                  ageHumanField.setText("72");
                  break;

               case 13:
                  ageHumanField.setText("77");
                  break;

               case 14:
                  ageHumanField.setText("82");
                  break;

               default:
                  ageHumanField.setText("That's a long life!");
            }

            return;

         }

         /* Creates a DogYearsFrame and makes it visible */
         public static void main(String[] args) {
            // Creates DogYearsFrame and its components
            DogYearsFrame myFrame = new DogYearsFrame();

            myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            myFrame.pack();
            myFrame.setVisible(true);

            return;
         }
      }
```

Screenshot:

The GUI consists of a JSpinner for selecting a dog's age and a non-editable text field that displays the converted age in human equivalent years. Visually, a JSpinner resembles a text field with two buttons on one end. Pressing either button allows the user to cycle through several values in fixed increments. In the example above, pressing the upper button increments the displayed value (i.e., the dog's age) by one year. Similarly, pressing the bottom button decrements the value by one year. The user may also type the desired value into the JSpinner. Values that either fall outside the expected range or do not conform to the expected formatting requirements are silently discarded, as is the case with a JFormattedTextField component.

Before creating a JSpinner component, a programmer must first determine the appropriate spinner model to use. A ***spinner model*** specifies the types of values that a corresponding JSpinner should expect and handle. A ***SpinnerNumberModel*** is a simple spinner model for representing a finite sequence of numerical values (e.g., 1, 2, and 3). The statement `spinnerModel = new SpinnerNumberModel(initYear, minYear, maxYear, stepVal);` creates a new SpinnerNumberModel object and assigns the SpinnerNumberModel object to the variable spinnerModel. The first three arguments within the parentheses specify the spinner's initial, minimum, and maximum values. For the above program, the initial age is 0, the minimum age is 0, and the maximum age is 30. The last argument within parentheses, i.e., stepVal, specifies the amount of each increment or decrement in value that will be made when the JSpinner's increment and decrement buttons are pressed. To use the SpinnerNumberModel class, the program imports the `javax.swing.SpinnerNumberModel;` class.

Note that a JSpinner can use a variety of different spinner models in order to represent sequences of Strings, dates, or even custom objects. Refer to How to Use Spinners from Oracle's Java tutorials for more information regarding other spinner models.

The statement `yearsSpinner = new JSpinner(spinnerModel);` creates a new JSpinner object and assigns its reference to the variable yearsSpinner. The item within parentheses specifies the spinner model. In this case, the variable spinnerModel refers to the aforementioned SpinnerNumberModel object. Using the JSpinner class requires including the import statement `import javax.swing.JSpinner;`.

A JSpinner generates ***change events*** that notify the underlying program of a change in the value displayed by the spinner. A ***ChangeListener object*** is used within a program to handle change

events, defining a stateChanged() method that dictates how the program should respond to a change event. The above program's DogYearsFrame class implements the ChangeListener interface, and thus defines the stateChanged() method. The statement `yearsSpinner.addChangeListener(this);` registers the current DogYearsFrame object as the spinner's designated change listener. Whenever the user types a new dog age or otherwise changes the displayed value (e.g., by increment or decrementing the value), DogYearsFrame's stateChanged() method extracts the displayed value via the statement `dogAgeYears = (Integer)yearsSpinner.getValue();`. To use both the ChangeEvent and ChangeListener class, a program must include the following import statements: `import javax.swing.event.ChangeEvent;` and `import javax.swing.event.ChangeListener;`.

Try 16.6.1: Allow non-integer values for a dog's age.

Modify the above program to allow the user to enter (or select) a dog's age in increments of 0.5. This requires changes to the spinner model and the stateChanged() method. In the stateChanged() method, make sure to use the appropriate data type for the spinner value (i.e., Double instead of Integer), and add more cases to the switch statement. You can use interpolation to determine the appropriate human age. For example, a dog age of 1.5 years results in a human age of (15 + 24) / 2 = 19.5 years.

In constructing the GUI's layout, notice that the DogYearsFrame() constructor utilizes two layout constraints: fill and anchor. Fill allows the programmer to specify the cardinal direction in which to resize a component. The fill value HORIZONTAL tells the layout manager to resize the component horizontally so that the component is wide enough to fill the cell. A fill value of VERTICAL tells the layout manager to resize the component in the vertical direction. The fill value BOTH stretches a component in both directions to fill the entire cell. The default value for the fill is NONE, which sizes a component according to the component's default preferred size. In the above program, the spinner and text field components use a horizontal fill so that they appear wider to the viewer.

The other layout constraint, anchor, allows the programmer to specify the location of a component within the component's containing cell. Some of the possible values are LINE_START, LINE_END, PAGE_START, PAGE_END, and the default CENTER. These anchor values place a component at the left, right, top, bottom, and center of the cell respectively. In the above program, the labels use an anchor value of LINE_END so that they are aligned on the right side of their containing cells and thus appear closer to the components which they describe.

For more information on layout constraints and possible values, refer to How to Use GridBagLayout from Oracle's Java tutorials.

P Participation Activity

16.6.1: Using JSpinners and spinner models.

#	Question	Your answer
1	Using a SpinnerNumberModel variable named digitModel, write a statement that creates a SpinnerNumberModel to represent the decimal digits (i.e., 0 through 9) with an initial value of 0.	
2	Write a statement that creates a JSpinner object and assigns it to the variable digitSpinner. Use digitModel as the spinner's model.	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials
- How to Use Spinners from Oracle's Java tutorials
- How to Use GridBagLayout from Oracle's Java tutorials

Section 16.7 - Displaying multi-line text in a JTextArea

A ***JTextArea*** is a Swing GUI component that supports the display of multiple lines of text. The following program uses a JTextArea to display the amount of money in a savings account per year based on the user-specified number of years, initial savings amount, and yearly interest rate.

Figure 16.7.1: Using a JTextArea to display the amount of money in a savings account per year.

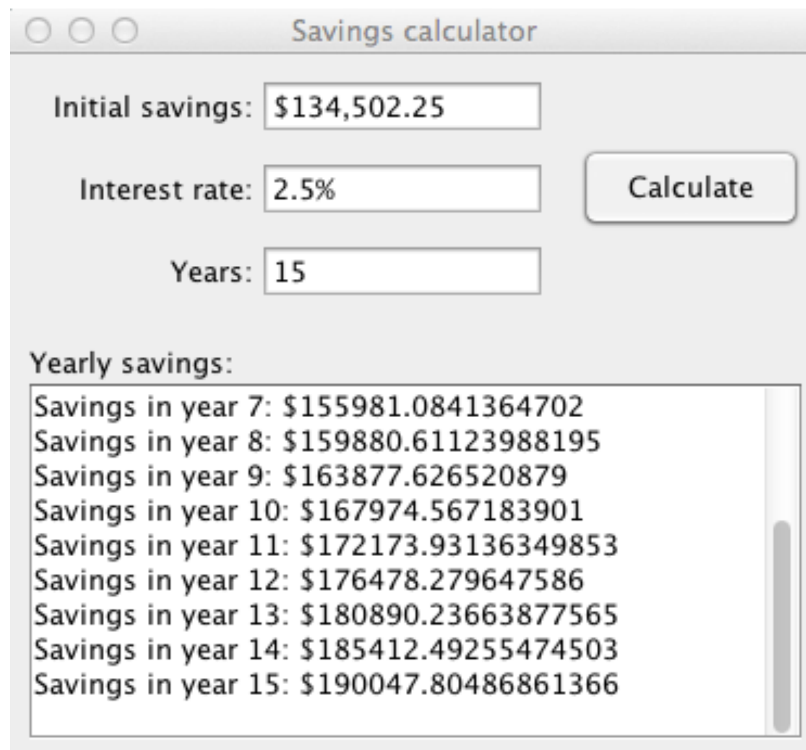
```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
```

```
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class SavingsInterestCalcFrame extends JFrame implements ActionListener {
   private JTextArea outputArea;                    // Displays yearly savings
   private JButton calcButton;                      // Triggers savings calculation
   private JFormattedTextField initSavingsField;    // Holds savings amount
   private JFormattedTextField interestRateField;  // Holds interest amount
   private JFormattedTextField yearsField;          // Holds num years

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   SavingsInterestCalcFrame() {
      GridBagConstraints layoutConst = null; // Used to specify GUI component layout
      JScrollPane scrollPane = null;          // Container that adds a scroll bar
      JLabel initSavingsLabel = null;         // Label for savings
      JLabel interestRateLabel = null;        // Label for interest
      JLabel yearsLabel = null;               // Label for num years
      JLabel outputLabel = null;              // Label for yearly savings

      // Format for the savings input
      NumberFormat currencyFormat = null;

      // Format for the interest rate input
      NumberFormat percentFormat = null;

      // Format for the years input
      NumberFormat integerFormat = null;

      // Set frame's title
      setTitle("Savings calculator");

      // Create labels
      initSavingsLabel = new JLabel("Initial savings:");
      interestRateLabel = new JLabel("Interest rate:");
      yearsLabel = new JLabel("Years:");
      outputLabel = new JLabel("Yearly savings:");

      // Create output area and add it to scroll pane
      outputArea = new JTextArea(10, 15);
      scrollPane = new JScrollPane(outputArea);
      outputArea.setEditable(false);

      calcButton = new JButton("Calculate");
      calcButton.addActionListener(this);

      // Create savings field and specify the currency format
      currencyFormat = NumberFormat.getCurrencyInstance();
      initSavingsField = new JFormattedTextField(currencyFormat);
      initSavingsField.setEditable(true);
      initSavingsField.setColumns(10); // Initial width of 10 units
      initSavingsField.setValue(0);
```

```
      // Create rate field and specify the percent format
      percentFormat = NumberFormat.getPercentInstance();
      percentFormat.setMinimumFractionDigits(1);
      interestRateField = new JFormattedTextField(percentFormat);
      interestRateField.setEditable(true);
      interestRateField.setValue(0.0);

      // Create years field and specify the default number (for doubles) format
      integerFormat = NumberFormat.getIntegerInstance();
      yearsField = new JFormattedTextField(integerFormat);
      yearsField.setEditable(true);
      yearsField.setValue(0);

      // Use a GridBagLayout
      setLayout(new GridBagLayout());

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 5, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
      add(initSavingsLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 5, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      add(initSavingsField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(5, 10, 5, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
      add(interestRateLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(5, 1, 5, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 1;
      add(interestRateField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(5, 10, 10, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 2;
      add(yearsLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(5, 1, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 2;
      add(yearsField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(0, 5, 0, 10);
      layoutConst.fill = GridBagConstraints.BOTH;
      layoutConst.gridx = 2;
```

```
      layoutConst.gridx = 2;
      layoutConst.gridy = 1;
      add(calcButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 1, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 3;
      add(outputLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 4;
      layoutConst.gridwidth = 3; // 3 cells wide
      add(scrollPane, layoutConst);
   }

   @Override
   public void actionPerformed(ActionEvent event) {
      int i = 0;                     // Loop index
      double savingsDollars = 0.0; // Yearly savings
      double interestRate = 0.0;   // Annual interest rate
      int numYears = 0;              // Num years to calc savings

      // Get values from fields
      savingsDollars = ((Number) initSavingsField.getValue()).intValue();
      interestRate = ((Number) interestRateField.getValue()).doubleValue();
      numYears = ((Number) yearsField.getValue()).intValue();

      // Clear the text area
      outputArea.setText("");

      // Calculate savings iteratively in a while loop
      i = 1;
      while (i <= numYears) {
         outputArea.append("Savings in year " + i +
                           ": $" + savingsDollars + "\n");
         savingsDollars = savingsDollars + (savingsDollars * interestRate);

         i = i + 1;
      }

      return;
   }

   /* Creates a SavingsInterestCalcFrame and makes it visible */
   public static void main(String[] args) {
      // Creates SavingsInterestCalcFrame and its components
      SavingsInterestCalcFrame myFrame = new SavingsInterestCalcFrame();

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.setVisible(true);

      return;
   }
}
```

Screenshot:



The GUI uses a large text area for displaying multiple lines of output (i.e., savings amount per year). The statement `outputArea = new JTextArea(10,15);` creates a JTextArea object and assigns it to the variable outputArea. The two literals within parentheses denote the dimensions (i.e., number of rows and columns) of the outputArea, therefore this particular text area has 10 rows and 15 columns. By default, a user can edit the text displayed by a JTextArea component. Because the above program only utilizes the text area for output, the program calls JTextArea's setEditable() method with a boolean argument of false, as in `outputArea.setEditable(false);`, in order to make the text area uneditable.

Note that the text area is functional after executing the previous statements. However, such an output area would not automatically scroll if the displayed text exceeded the dimensions of the component. In order to enable scrolling, a programmer must add a JTextArea component to a JScrollPane. A ***JScrollPane*** is a Swing GUI component that provides a scrollable view to the underlying component JScrollPane manages, also called a client. The statement `scrollPane = new JScrollPane(outputArea);` assigns a new JScrollPane object to the variable scrollPane. The argument within parentheses specifies the scroll pane's client, which in this case corresponds to the outputArea JTextArea object.

Importantly, the statement `add(scrollPane, layoutConst);` adds the scrollPane, not the outputArea, to the frame. A common error is to add both a JScrollPane object and the JScrollPane's client to a frame, resulting in a GUI with both an empty scroll pane and the client component. In order to make the scrollPane and scrollPane's client wider, the statement

`layoutConst.gridwidth = 3;` sets the scrollPane's layout width to 3 cells. A width of 3 cells also nicely aligns the scrollPane with the labels, fields, and button above.

Finally, the program sets the SavingsInterestCalcFrame object as calcButton's ActionListener. Thus, pressing the GUI's "Calculate" button invokes the above actionPerformed() method, which extracts the inputs from the three JFormattedTextField components and uses a while loop to iteratively calculate the user's savings every year. The program uses the statement `outputArea.setText("");` to overwrite all of the text in the text area with an empty string, thereby clearing the contents of the text area. The program then displays the user's savings for each year using the JTextArea's append() method. As the method's name implies, the append() method appends the String argument at the end of the text area. Note that the append() method does not insert newline characters automatically; that's the programmer's responsibility.

P Participation Activity

16.7.1: Creating and using scrollable text areas.

#	Question	Your answer
1	Write a statement that creates a JTextArea component with 20 rows and 30 columns. Assign the JTextArea object to a variable called displayArea.	
2	Write a statement that creates a JScrollPane object with the client specified by the displayArea variable. Assign the JScrollPane object to a variable called myScroller.	
3	Write a statement that appends the String literal "Done!" to the end of a JTextArea component whose variable is called statusOutput.	
4	Consider the above myScroller and displayArea components. Should the programmer add both components to the frame individually (Answer Yes or No)?	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials

Section 16.8 - Using tables in GUIs

Tables are convenient structures for organizing and displaying information. A ***JTable*** is a Swing GUI component that displays data in a table, optionally allowing the GUI user to edit the data by entering new values into the table. The following program finds the maximum integer value within a user-specified array, whose elements are displayed using an editable JTable.

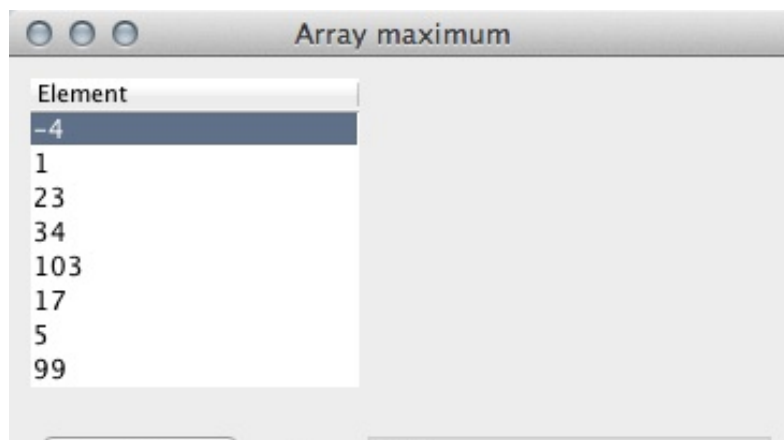
Figure 16.8.1: Calculating the maximum array value for an array displayed in a JTable.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTable;

public class ArrayMaxFrame extends JFrame implements ActionListener {
   private JLabel maxLabel;                  // Label for max array element
   private JFormattedTextField maxField; // Holds max array element
   private JButton maxButton;                // Triggers search for max array element
   private JTable arrayValsTable;            // Table of array values
   private final int numElements = 8;        // Number of array elemeents
   private String[] columnHeadings;          // Stores the table's column headings
   private String[][] tableVals;             // Stores the table's values

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   ArrayMaxFrame() {
      GridBagConstraints layoutConst = null; // GUI component layout
      int i = 0;

      // Set frame's title
      setTitle("Array maximum");

      // Create label
      maxLabel = new JLabel("Max:");
```

```
      // Create field
      maxField = new JFormattedTextField(NumberFormat.getIntegerInstance());
      maxField.setColumns(15);
      maxField.setEditable(false);
      maxField.setValue(0);

      // Create button
      maxButton = new JButton("Find max");
      maxButton.addActionListener(this);

      // Table headings and values
      columnHeadings = new String[1];
      tableVals = new String[8][1];

      // Initialize column heading(s)
      columnHeadings[0] = "Element";

      // Initialize table values
      for (i = 0; i < numElements; ++i) {
         tableVals[i][0] = "0";
      }

      // Create a table with the specified values and column headings
      arrayValsTable = new JTable(tableVals, columnHeadings);

      // Use a GridBagLayout
      setLayout(new GridBagLayout());

      // Add table header
      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 0, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
      layoutConst.gridwidth = 2;
      add(arrayValsTable.getTableHeader(), layoutConst);

      // Add table itself
      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(0, 10, 10, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
      layoutConst.gridwidth = 2;
      add(arrayValsTable, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 2;
      add(maxButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 1;
      layoutConst.gridy = 2;
      add(maxLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 10, 10);
```

```
         layoutConst.insets = new Insets(10, 1, 10, 10);
         layoutConst.fill = GridBagConstraints.HORIZONTAL;
         layoutConst.gridx = 2;
         layoutConst.gridy = 2;
         add(maxField, layoutConst);
      }

      @Override
      public void actionPerformed(ActionEvent event) {
         int i = 0;             // Loop index
         int maxElement = 0;   // Max value found
         String strElem = ""; // Array element value (string)
         int elemVal = 0;      // Array element value (int)

         strElem = tableVals[0][0];                // Get table value (String)
         maxElement = Integer.parseInt(strElem); // Convert to Integer

         // Iterate through table values to find max
         for (i = 1; i < numElements; ++i) {
            strElem = tableVals[i][0];             // Get table value (String)
            elemVal = Integer.parseInt(strElem); // Convert to Integer
            if (elemVal > maxElement) {            // Check if new max value found
               maxElement = elemVal;               // If so, update max
            }
         }

         // Display maximum value
         maxField.setValue(maxElement);

         return;
      }

      /* Creates a ArrayMaxFrame and makes it visible */
      public static void main(String[] args) {
         // Creates ArrayMaxFrame and its components
         ArrayMaxFrame myFrame = new ArrayMaxFrame();

         myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         myFrame.pack();
         myFrame.setVisible(true);

         return;
      }
   }
```

Screenshot:



Find max Max: 103

The program uses the array tableVals to store the elements that the user edits. Although one would expect to use a one-dimensional array of integer values, notice that tableVals is a two-dimensional array of String elements. Because a table typically consists of a two-dimensional grid of table elements, also known as table cells, a programmer may only display elements stored in a two-dimensional array in order to ensure a direct mapping between an array element's index to a table's cell index (e.g., the element given by `tableVals[i][j];` is associated with the table cell in the ith row and jth column). Thus, tableVals is initialized as a two-dimensional array with 8 rows and 1 column, as in `tableVals = new String[8][1];`, instead of a one-dimensional array with 8 elements.

By default, a JTable uses a simple ***table model***, which is an object responsible for managing the table's data. The table model interprets all table cell values using a String representation. The above program stores the array elements as Strings and converts these elements to integers, when necessary, by using the Integer class's parseInt() method (e.g., `max = Integer.parseInt(strElem);`).

Note that the default table model can actually display array elements of any reference data type (e.g., Integer, Double) because all reference types implicitly define the toString() method, which returns a String representation of the object. However, the default table model does not perform this conversion in the other direction. In other words, the table model does not convert a cell's text back into the appropriate data type.

After properly initializing the data array, the program creates a JTable object and assigns the object to the variable arrayValsTable via the statement `arrayValsTable = new JTable(tableVals, columnHeadings);`. The first argument within parentheses, i.e., tableVals, corresponds to the array that the programmer wants to display within the table. The second argument, i.e., columnHeadings, is a one-dimensional String array that contains the names, or headings, of each column. Table headings are contained in a separate area of the table known as the ***header***. The columnHeadings array, for example, contains a single String, "Element", corresponding to the heading of the table's only column. In order to use a JTable, the program must include the import statement `import javax.swing.JTable;`.

A JTable requires the programmer to specify separate layout constraints (i.e., position, size, etc.) for the table's header and the table's cells. The above program uses a GridBagLayout, as usual, to place the header in the grid cell just above the table's cells. JTable's getTableHeader() method returns a reference to the header and the statement `add(arrayValsTable.getTableHeader(), layoutConst);` uses this method to add the table's header to the frame using the constraints specified by layoutConst. A programmer can then add the table's cells to a frame by using the JTable variable on its own, as in `add(arrayValsTable, layoutConst);`. One alternative that does not require separate

treatment of the table's header and cells is to use a JScrollPane as that table's container, as in `JScrollPane scrollPane = new JScrollPane(arrayValsTable);`, and then add the JScrollPane to the frame. Lastly, a programmer can also choose not to add a table's header to the frame, which results in a valid GUI that only displays the table's cells.

Try 16.8.1: Finding the maximum value in a two-dimensional array.

Modify the above ArrayMaxFrame program so that it finds the maximum value within a user-specified **two-dimensional** array -- i.e., the array should have 8 rows and 2 columns. Necessary changes include specifying the appropriate size for the tableVals array and modifying the for loop in the actionPerformed() method to iterate through all elements.

Participation Activity

16.8.1: Creating and using a JTable.

For the following statements, assume that all JTable objects use a default table model and are initialized using an array.

#	Question	Your answer
1	A programmer can display an Integer array such as `Integer[][] tableVals;` in an editable JTable and expect the user to be able to successfully edit the elements within the array.	True False
2	A JTable's header can be added to a separate grid cell when using a GridBagLayout layout manager.	True False
3	When the user edits a table's cell(s), the array associated with the table is also updated.	True False

Exploring further:

- How to use various Swing components from Oracle's Java tutorials

---

Section 16.9 - Using sliders in GUIs

A ***JSlider*** is a Swing GUI component that allows users to select a numeric value from within a predefined range. For example, the following GUI program uses a JSlider component to allow the user to enter a person's height in U.S. units (feet and inches). The program then converts the input height into centimeters.

Figure 16.9.1: Using JSliders to enter height in feet and inches.

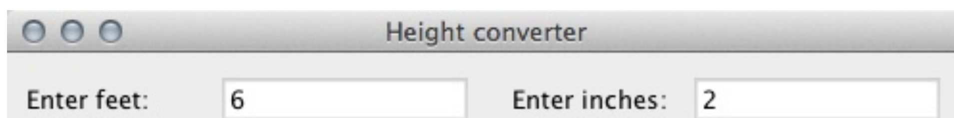
```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class HeightConverterFrame extends JFrame implements ActionListener, ChangeLi
   private JTextField heightCmField;  // Holds height output value in cm
   private JTextField heightFtField;  // Holds height input value in feet
   private JTextField heightInField;  // Holds heigth input value in inches
   private JLabel feetLabel;          // Label for heigth input in feet
   private JLabel inchesLabel;        // Label for height input in inches
   private JLabel cmLabel;            // Label for heigth in cm
   private JButton convertButton;     // Triggers heigth conversion
   private JSlider heightFtSlider;    // Slider for feet input
   private JSlider heightInSlider;    // Slider for inches input

   final static double CM_PER_IN = 2.54; // Centimeters per inch
   final static int IN_PER_FT = 12;      // Inches per foot

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   HeightConverterFrame() {
      int feetMin = 0;     // Feet slider min value
      int feetMax = 10;    // Feet slider max value
      int feetInit = 5;    // Feet slider initial value
      int inchesMin = 0;   // Inches slider min value
      int inchesMax = 11;  // Inches slider max value
      int inchesInit = 8;  // Inches sldier initial value
```

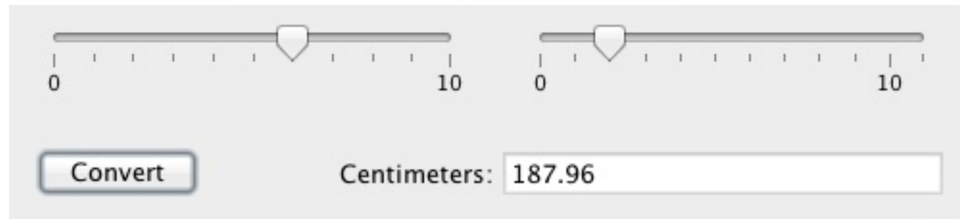
```
        GridBagConstraints layoutConst = null; // GUI component layout

        // Set frame's title
        setTitle("Height converter");

        // Create labels
        feetLabel = new JLabel("Enter feet:");
        inchesLabel = new JLabel("Enter inches:");
        cmLabel = new JLabel("Centimeters:");

        heightCmField = new JTextField(10);
        heightCmField.setEditable(false);

        convertButton = new JButton("Convert");
        convertButton.addActionListener(this);

        // Create slider that enables user to enter height in feet
        heightFtSlider = new JSlider(feetMin, feetMax, feetInit);
        heightFtSlider.addChangeListener(this); // Use HeightConverter's stateChanged(
        heightFtSlider.setMajorTickSpacing(10);
        heightFtSlider.setMinorTickSpacing(1);
        heightFtSlider.setPaintTicks(true);
        heightFtSlider.setPaintLabels(true);

        heightFtField = new JTextField(10);
        heightFtField.setEditable(false);
        heightFtField.setText("5");

        // Creates slider that enables user to enter height in inches
        heightInSlider = new JSlider(inchesMin, inchesMax, inchesInit);
        heightInSlider.addChangeListener(this); // Use HeightConverter's stateChanged(
        heightInSlider.setMajorTickSpacing(10);
        heightInSlider.setMinorTickSpacing(1);
        heightInSlider.setPaintTicks(true);
        heightInSlider.setPaintLabels(true);

        heightInField = new JTextField(10);
        heightInField.setEditable(false);
        heightInField.setText("8");

        // Create frame and add components using GridBagLayout
        setLayout(new GridBagLayout());

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 10, 1, 1);
        layoutConst.anchor = GridBagConstraints.LINE_START;
        layoutConst.gridx = 0;
        layoutConst.gridy = 0;
        layoutConst.gridwidth = 1;
        add(feetLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 10, 1, 1);
        layoutConst.anchor = GridBagConstraints.LINE_START;
        layoutConst.gridx = 2;
        layoutConst.gridy = 0;
        layoutConst.gridwidth = 1;
        add(inchesLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 1, 1, 10);
```

```
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      layoutConst.gridwidth = 1;
      add(heightFtField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 1, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 3;
      layoutConst.gridy = 0;
      layoutConst.gridwidth = 1;
      add(heightInField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
      layoutConst.gridwidth = 2;
      add(heightFtSlider, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 2;
      layoutConst.gridy = 1;
      layoutConst.gridwidth = 2;
      add(heightInSlider, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 5);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 0;
      layoutConst.gridy = 2;
      layoutConst.gridwidth = 1;
      add(convertButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 10, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 1;
      layoutConst.gridy = 2;
      layoutConst.gridwidth = 1;
      add(cmLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 2;
      layoutConst.gridy = 2;
      layoutConst.gridwidth = 2;
      add(heightCmField, layoutConst);
   }

   /* Converts a height in feet/inches to centimeters. */
   public static double HeightFtInToCm(int ft, int in) {
      int totIn = 0;           // Total inches input by user
      double cmHeight = 0.0;   // Corresponding height in cm

      totIn = (ft * IN_PER_FT) + in; // Total inches
      cmHeight = totIn * CM_PER_IN;   // Conver to cm
```

```
      cmHeight = totIn * CM_PER_IN;  // Conver to cm
      return cmHeight;
   }

   /* Called as slider value changes. Updates fields to display
      the numerical representation of the slider settings. */
   @Override
   public void stateChanged(ChangeEvent event) {
      int sliderVal = 0;          // Slider value (int)
      String strSliderVal = ""; // Slider value (string)

      // Get source of event (2 sliders in GUI)
      JSlider sourceEvent = (JSlider) event.getSource();

      if (sourceEvent == heightFtSlider) {
         sliderVal = heightFtSlider.getValue();      // Get slider value
         strSliderVal = Integer.toString(sliderVal); // Conver to int
         heightFtField.setText(strSliderVal);        // Update display
      }
      else if (sourceEvent == heightInSlider) {
         sliderVal = heightInSlider.getValue();
         strSliderVal = Integer.toString(sliderVal);
         heightInField.setText(strSliderVal);
      }

   }

   /* Called when button is pressed. Converts height to cm. */
   @Override
   public void actionPerformed(ActionEvent event) {
      int ftVal = 0;       // User defined height in feet
      int inVal = 0;       // User defined height in inches
      double cmVal = 0.0; // Corresponding height in cm

      ftVal = heightFtSlider.getValue();    // Get ft slider value
      inVal = heightInSlider.getValue();    // Get in slider vlaue
      cmVal = HeightFtInToCm(ftVal, inVal); // Convert ft/in to cm, update cmVal

      heightCmField.setText(Double.toString(cmVal)); // Update cm heigth

      return;
   }

   /* Creates a HeightConverterFrame and makes it visible */
   public static void main(String[] args) {
      // Creates HeightConverterFrame and its components
      HeightConverterFrame myFrame = new HeightConverterFrame();

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.setVisible(true);

      return;
   }
}
```

Screenshot:





The above program uses two JSliders for the input height: one for feet and another for inches. The user is able to drag each slider's knob in order to select a desired value, which the program displays in a non-editable text field for added clarity. The convert button triggers the actionPerformed() method, which performs the height conversion and displays the final height value in a text field.

To use the JSlider class, the program must include the import statement `import javax.swing.JSlider;`. The statement `heightFtSlider = new JSlider(feetMin, feetMax, feetInit);` assigns a newly created JSlider object to the variable heightFtSlider, with the arguments feetMin, feetMax, and feetInit specifying the slider's minimum, maximum, and initial values respectively. The heightFtSlider slider can represent values between 0 and 10 feet, and the heightInSlider slider allows inputs between 0 and 11 inches.

By default, JSlider components do not show tick marks. The JSlider class's setMajorTickSpacing() method allows the programmer to specify the value spacing between consecutive tick marks as an integer argument. For example, the statement `heightFtSlider.setMajorTickSpacing(10);` configures the heightFtSlider slider to display major tick marks every 10 feet. Similarly, JSlider's setMinorTickSpacing() method allows the programmer to specify the spacing between minor tick marks, as in the statement `heightFtSlider.setMinorTickSpacing(1);`, which sets minor tick marks at every foot value. Minor and major tick marks differ solely in their displayed size, with major tick marks being larger than minor tick marks. After specifying tick mark spacings, the programmer must invoke JSlider's setPaintTicks() with the boolean literal true as an argument in order to show the tick marks. Additionally, JSlider's setPaintLabels() method takes a boolean value as an argument in order to allow the programmer to specify whether the JSlider component should display a value at every major tick. Thus, the statement `heightInSlider.setPaintLabels(true);` tells the GUI to display the values at every major tick mark, i.e., every 10 feet.

Try 16.9.1: Labeling every foot and inch value.

Modify the above HeightConverterFrame program so that both sliders display the value at every integer value (i.e., 1, 2, 3, etc.). Recall that JSlider's setPaintLabels() method only allows the programmer to display values at major tick marks, thus, one option is to remove all minor tick marks and change the spacing between major tick marks to one.

The HeightConverterFrame class implements both an ActionListener and a ChangeListener in order to detect events from the button and sliders respectively. Both JSlider components register the current HeightConverterFrame object as the designated change listener. Thus, the program invokes the stateChanged() method whenever the user selects a height value using either slider. Because the stateChanged() method handles change events from two different sources (i.e., both sliders), the method first gets the source of the event via ChangeEvent's getSource() method, as in `JSlider sourceEvent = (JSlider) event.getSource();`, and stores the returned reference to the source component in the local JSlider variable called sourceEvent. Next, the stateChanged() method compares the source with the slider variables heightFtSlider and heightInSlider in order to determine which slider the user changed. Once the method determines the source component, the method uses JSlider's getValue() method to extract the slider's value, `sliderVal = heightFtSlider.getValue();`, and then displays the value in the appropriate text field. Thus, the program can dynamically update the heightFtField and heightInField text fields with the current values entered into the heightFtSlider and heightInSlider sliders, respectively.

P Participation Activity

16.9.1: Creating and using sliders.

#	Question	Your answer
1	Write a statement that creates a JSlider component with a minimum value of -10, a maximum value of 10, and an initial value of 0. Assign the JSlider object to a variable called locationSlider.	
2	Write two statements that first set locationSlider's minor tick spacing to 1, and then set locationSlider's major tick spacing to 5.	
3	Write two statements that first configure locationSlider to display all tick marks and then configure locationSlider so that it **does not** display labels at major tick marks.	
4	Write a statement that gets locationSlider's current value and stores it into a variable called locVal.	

Exploring further:

- How to use various Swing components from Oracle's Java tutorials

Section 16.10 - GUI tables, fields, and buttons: A seat reservation example

The following program combines a table, fields, buttons, and dialogs to create a GUI that allows a reservation agent to reserve seats for people, as might be useful for a theater or an airplane.

Figure 16.10.1: A seat reservation GUI involving a table, fields, and buttons.

SeatInfo.java

```
public class SeatInfo {
   private String firstName; // First name
   private String lastName;  // Last name
   private int amtPaid;      // Amount paid

   // Method to initialize Seat fields
   public void reserveSeat(String inFirstName, String inLastName, int ticketCost) {
      firstName = inFirstName;
      lastName = inLastName;
      amtPaid = ticketCost;
      return;
   }

   // Method to empty a Seat
   public void makeEmpty() {
      firstName = "empty";
      lastName = "empty";
      amtPaid = 0;
      return;
   }

   // Method to check if Seat is empty
   public boolean isEmpty() {
      return firstName.equals("empty");
   }

   // Method to print Seat fields
   public void printSeatInfo() {
      System.out.print(firstName + " ");
      System.out.print(lastName + " ");
      System.out.println("Paid: " + amtPaid);
      return;
   }

   public String getFirstName() {
      return firstName;
   }

   public String getLastName() {
      return lastName;
   }

   public int getAmountPaid() {
      return amtPaid;
   }
}
```

SeatReservationFrame.java

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import java.util.ArrayList;
```

```
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import javax.swing.JTextField;

public class SeatReservationFrame extends JFrame implements ActionListener {
   private JTextField firstNameField;             // Holds first name
   private JTextField lastNameField;              // Holds last name
   private JFormattedTextField seatNumField;      // Holds seat number
   private JFormattedTextField amountPaidField;   // Holds ticket cost
   private JLabel tableLabel;                     // Label for table display
   private JLabel seatNumLabel;                   // Label for seat number
   private JLabel firstNameLabel;                 // Label for first name
   private JLabel lastNameLabel;                  // Label for last name
   private JLabel amountPaidLabel;                // Label for amount paid
   private JButton reserveButton;                 // Triggers seat reservation
   private JButton quitButton;                    // Triggers termination of GUI
   private JTable seatStatusTable;                // Table tracks seat reservations
   private final static int NUM_SEATS = 5;        // Number of seat in reservation s
   private static ArrayList<SeatInfo> seatResArr; // ArrayList of Seat objects

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   SeatReservationFrame() {
      Object[][] tableVals = new Object[5][4];                 // Seat reservation ta
      String[] columnHeadings = {"Seat Number", "First Name", // Column headings for
                                 "Last Name", "Amount Paid"};
      GridBagConstraints layoutConst = null;                   // GUI component layou
      NumberFormat currencyFormat = null;                      // Format for amount p

      // Set frame's title
      setTitle("Seat reservation");

      // Add 5 seat objects to ArrayList
      seatResArr = new ArrayList<SeatInfo>();
      seatsAddElements(seatResArr, NUM_SEATS);

      // Make all seats empty
      seatsMakeEmpty(seatResArr);

      // Create seat reservation table
      tableLabel = new JLabel("Seat reservation status:");
      seatNumLabel = new JLabel("Seat Number:");
      firstNameLabel = new JLabel("First Name:");
      lastNameLabel = new JLabel("Last Name:");
      amountPaidLabel = new JLabel("Amount Paid:");

      seatNumField = new JFormattedTextField(NumberFormat.getIntegerInstance());
      seatNumField.setEditable(true);
      seatNumField.setValue(0);

      firstNameField = new JTextField(20);
      firstNameField.setEditable(true);
      firstNameField.setText("John");

      lastNameField = new JTextField(20);
      lastNameField.setEditable(true);
      lastNameField.setText("Doe");
```

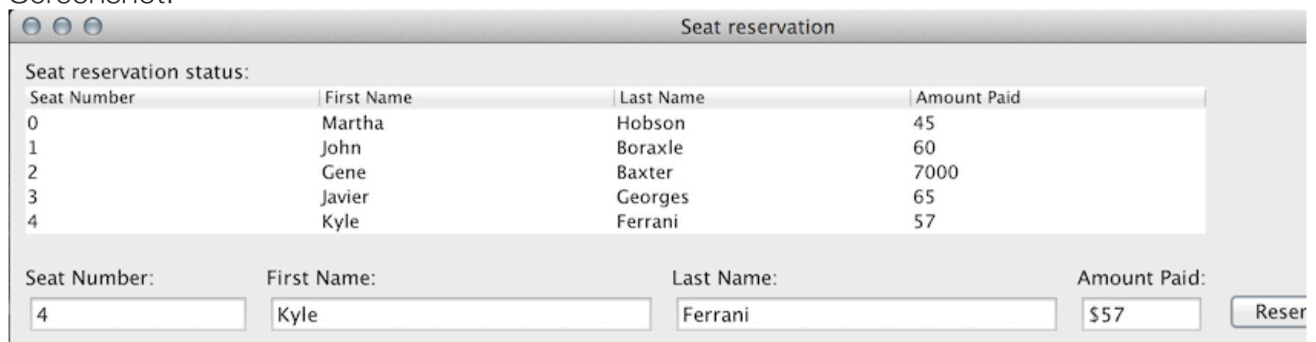
```
        currencyFormat = NumberFormat.getCurrencyInstance();
        currencyFormat.setMaximumFractionDigits(0);
        amountPaidField = new JFormattedTextField(currencyFormat);
        amountPaidField.setEditable(true);
        amountPaidField.setValue(0.0);

        reserveButton = new JButton("Reserve");
        reserveButton.addActionListener(this);

        quitButton = new JButton("Quit");
        quitButton.addActionListener(this);

        // Initialize table
        seatStatusTable = new JTable(tableVals, columnHeadings);
        seatStatusTable.setEnabled(false); // Prevent user input via table

        // Add components using GridBagLayout
        setLayout(new GridBagLayout());

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 10, 1, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 0;
        layoutConst.gridy = 0;
        add(tableLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(1, 10, 0, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 0;
        layoutConst.gridy = 1;
        layoutConst.gridwidth = 4;
        add(seatStatusTable.getTableHeader(), layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(0, 10, 10, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 0;
        layoutConst.gridy = 2;
        layoutConst.gridwidth = 4;
        add(seatStatusTable, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 10, 1, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 0;
        layoutConst.gridy = 3;
        add(seatNumLabel, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(1, 10, 10, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 0;
        layoutConst.gridy = 4;
        add(seatNumField, layoutConst);

        layoutConst = new GridBagConstraints();
        layoutConst.insets = new Insets(10, 10, 1, 0);
        layoutConst.fill = GridBagConstraints.HORIZONTAL;
        layoutConst.gridx = 1;
        layoutConst.gridy = 3;
```

```
      layoutConst.gridy = 3;
      add(firstNameLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 1;
      layoutConst.gridy = 4;
      add(firstNameField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 1, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 2;
      layoutConst.gridy = 3;
      add(lastNameLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 2;
      layoutConst.gridy = 4;
      add(lastNameField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 1, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 3;
      layoutConst.gridy = 3;
      add(amountPaidLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 3;
      layoutConst.gridy = 4;
      add(amountPaidField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(0, 10, 10, 5);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 4;
      layoutConst.gridy = 4;
      add(reserveButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(0, 5, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 5;
      layoutConst.gridy = 4;
      add(quitButton, layoutConst);
   }

   /* Called when either button is pressed. */
   @Override
   public void actionPerformed(ActionEvent event) {
      SeatInfo seatElement;  // Seat information
      String firstName = ""; // First name
      String lastName = "";  // Last name
      int seatNum = 0;       // Seat number
      int amtPaid = 0;       // Amount paid
```

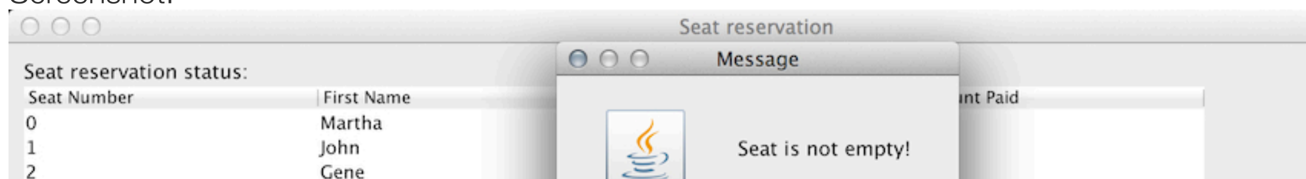
```
      // Get source of event (2 buttons in GUI)
      JButton sourceEvent = (JButton) event.getSource();

      // User pressed the reserve button
      if (sourceEvent == reserveButton) {
         seatNum = ((Number) seatNumField.getValue()).intValue();

         // User tried to reserve non-existing seat
         if (seatNum >= NUM_SEATS) {
            // Show failure dialog
            JOptionPane.showMessageDialog(this, "Seat doesn't exist!");
         }
         // User tried to reserve a non-empty seat
         else if (!(seatResArr.get(seatNum).isEmpty())) {
            // Show failure dialog
            JOptionPane.showMessageDialog(this, "Seat is not empty!");
         }
         // Reserve the specified seat
         else {
            firstName = firstNameField.getText();
            lastName = lastNameField.getText();
            amtPaid = ((Number) amountPaidField.getValue()).intValue();

            seatElement = new SeatInfo();          // Create new Seat object
            seatElement.reserveSeat(firstName, lastName, amtPaid);
            seatResArr.set(seatNum, seatElement); // Add seat to ArrayList

            updateTable();                         // Synchronize table with sts Arra

            // Show success dialog
            JOptionPane.showMessageDialog(this, "Seat reservation completed.");
         }
      }
      else if (sourceEvent == quitButton) {
         dispose();                                // Terminate program
      }

      return;
   }

   /* Updates the reservation information displayed by the table */
   public void updateTable() {
      final int seatNumCol = 0;   // Col num for seat numbers
      final int firstNameCol = 1; // Col num for first names
      final int lastNameCol = 2;  // Col num for last names
      final int paidCol = 3;      // Col num for amount paid
      int i = 0;                  // Loop index

      for (i = 0; i < NUM_SEATS && i < seatResArr.size(); ++i) {
         if (seatResArr.get(i).isEmpty()) { // Clear table entries
            seatStatusTable.setValueAt(null, i, seatNumCol);
            seatStatusTable.setValueAt(null, i, firstNameCol);
            seatStatusTable.setValueAt(null, i, lastNameCol);
            seatStatusTable.setValueAt(null, i, paidCol);
         }
         else {                             // Update table with content in the seat
            seatStatusTable.setValueAt(i, i, seatNumCol);
            seatStatusTable.setValueAt(seatResArr.get(i).getFirstName(), i, firstNam
            seatStatusTable.setValueAt(seatResArr.get(i).getLastName(), i, lastNameC
            seatStatusTable.setValueAt(seatResArr.get(i).getAmountPaid(), i, paidCol
         }
```

```
      }

      return;
   }

   /* Makes seats empty */
   public static void seatsMakeEmpty(ArrayList<SeatInfo> seatsRes) {
      int i = 0; // Loop index

      for (i = 0; i < seatsRes.size(); ++i) {
         seatsRes.get(i).makeEmpty();
      }
      return;
   }

   /* Adds empty seats to ArrayList */
   public static void seatsAddElements(ArrayList<SeatInfo> seatsRes, int numSeats) {
      int i = 0; // Loop index

      for (i = 0; i < numSeats; ++i) {
         seatsRes.add(new SeatInfo());
      }
      return;
   }

   /* Creates a SeatReservationFrame and makes it visible */
   public static void main(String[] args) {
      // Creates SeatReservationFrame and its components
      SeatReservationFrame myFrame = new SeatReservationFrame();

      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.pack();
      myFrame.setVisible(true);

      return;
   }
}
```

Screenshot:



Screenshot:



3 Javier
4 Kyle
OK
Seat Number: First Name: Amount Paid:
4 John Walker $57 Reser

The above program defines a SeatInfo class to store information regarding a particular reservation. The SeatInfo class defines fields for a person's first name, last name, and the amount paid for the particular seat. The SeatInfo class also defines methods that allow a programmer to reserve a seat, check if a seat is empty, or make the seat empty.

The program creates an ArrayList of 5 Seat objects called seatResArr, which represents, for example, the entire theater or airplane. The program initializes all seats to empty, as indicated by a first and last name of "empty", and then allows the user to reserve a seat by entering the required seat information into the appropriate text fields and pressing the "Reserve" button. The table then displays information for each seat in a separate row.

The SeatReservationFrame class defines several methods that iterate through an ArrayList of SeatInfo objects in order to perform useful operations. The seatsAddElements() method takes an empty ArrayList and adds the desired number of seats. The seatsMakeEmpty() method iterates through an ArrayList in order to make all seats empty.

The GUI's two buttons, whose references variables are reserveButton and quitButton, use the current SeatReservationFrame object as the ActionListener. Thus, the program calls the actionPerformed() method when the user presses the reserveButton to make a seat reservation or when the user presses the quitButton to terminate the program. The actionPerformed() method first determines the source component of the action event by using ActionEvent's getSource() method, which returns a reference to the object that triggered the event, and compares the returned reference to reserveButton and quitButton. If the user pressed the quitButton, then the actionPerformed() method calls JFrame's dispose() method to terminate the program and GUI. Otherwise, the actionPerformed() method attempts to reserve a seat, executing a series of checks to ensure the user entered valid reservation information.

If the user tries to reserve a non-existing seat, the actionPerformed() method displays a dialog window with the message "Seat doesn't exist!". If instead the user tried to reserve a seat that is not currently empty, the program displays a dialog window with the message "Seat is not empty!". Otherwise, the method extracts the seat number, first name, last name, and amount paid from the corresponding text fields, creates a new SeatInfo object with the provided information, adds the SeatInfo object to the seatResArr ArrayList, and calls the UpdateTable() method to update the table with the new reservation data.

Try 16.10.1: Modify the above GUI.

Modify the above SeatReservationFrame program to have an additional JFormattedTextField and JButton component for the purposes of deleting a particular seat reservation. The JFormattedTextField component should allow the user to enter the seat number that should be deleted, and the JButton should trigger the deletion.

Notice that the program stores JTable values in an array of elements of type Object instead of String, as seen in previous examples. Because all classes in Java are also of type Object, a programmer can exploit an advanced concept known as polymorphism to store elements of various reference types in a single array of Object elements. For example, the tableVals array is defined as a two-dimensional array of Objects because it needs to store elements of type String (e.g., first name, last name, etc.) and elements of type Number (e.g., seat number and payment amount).

P Participation Activity

16.10.1: Multiple action event sources and tables.

#	Question	Your answer
1	An ActionListener object can only handle events from one source (e.g., a single button).	True False
2	The ActionEvent class's getSource() method returns a reference to the object (e.g., an instance of JButton) on which the event occurred.	True False
3	A JTable can display elements of different reference types.	True False

Section 16.11 - Reading files with a GUI

A ***JFileChooser*** is a Swing GUI component that supports directory navigation and file selection. The following example presents a GUI that allows the user to select a file with a JFileChooser component and then prints the file's contents as Unicode characters.

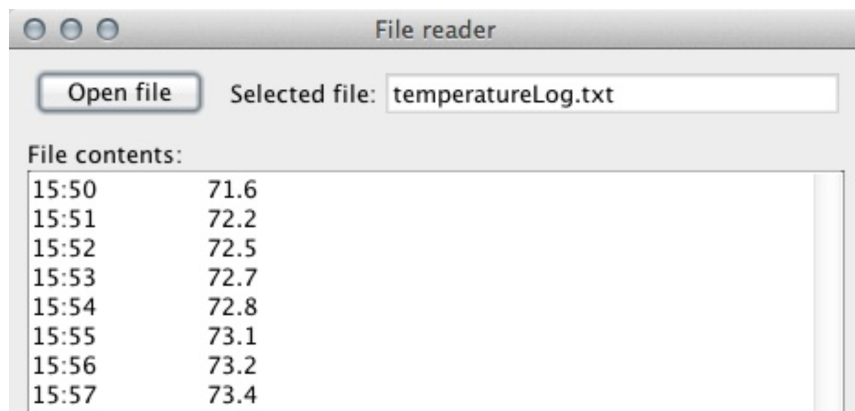
Figure 16.11.1: Using a JFileChooser to select a file for reading.

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class FileReadFrame extends JFrame implements ActionListener {
   private JScrollPane scrollPane;        // Container adds scroll bar
   private JTextArea outputArea;          // Holds file output
   private JLabel selectedFileLabel;      // Label for file name
   private JLabel outputLabel;            // Label for file contents
   private JTextField selectedFileField; // Holds name of file
   private JFileChooser fileChooser;      // Enables user to select file
   private JButton openFileButton;        // Trigger file open

   /* Constructor creates GUI components and adds GUI components
      using a GridBagLayout. */
   FileReadFrame() {
      GridBagConstraints layoutConst = null; // GUI component layout

      // Set frame's title
      setTitle("File reader");

      outputLabel = new JLabel("File contents:");
      selectedFileLabel = new JLabel("Selected file:");

      selectedFileField = new JTextField(20);
      selectedFileField.setEditable(false);
      selectedFileField.setText("...");

      outputArea = new JTextArea(10, 25);
      scrollPane = new JScrollPane(outputArea);
      outputArea.setEditable(false);

      openFileButton = new JButton("Open file");
      openFileButton.addActionListener(this);
```

```
      // Create file chooser. It's not added to this frame.
      fileChooser = new JFileChooser();

      // Add components using GridBagLayout
      setLayout(new GridBagLayout());

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 10, 5, 5);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 0;
      add(openFileButton, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 5, 5, 1);
      layoutConst.anchor = GridBagConstraints.LINE_END;
      layoutConst.gridx = 1;
      layoutConst.gridy = 0;
      add(selectedFileLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(10, 1, 5, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 2;
      layoutConst.gridy = 0;
      layoutConst.gridwidth = 2;
      layoutConst.gridheight = 1;
      add(selectedFileField, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(5, 10, 0, 0);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 1;
      add(outputLabel, layoutConst);

      layoutConst = new GridBagConstraints();
      layoutConst.insets = new Insets(1, 10, 10, 10);
      layoutConst.fill = GridBagConstraints.HORIZONTAL;
      layoutConst.gridx = 0;
      layoutConst.gridy = 2;
      layoutConst.gridheight = 2;
      layoutConst.gridwidth = 4;
      add(scrollPane, layoutConst);
   }

   /* Called when openFileButton is pressed. */
   @Override
   public void actionPerformed(ActionEvent event) {
      FileInputStream fileByteStream = null; // File input stream
      Scanner inFS = null;                   // Scanner object
      String readLine = "";                  // Input from file
      File readFile = null;                  // Input file
      int fileChooserVal = 0;                // File chooser

      // Open file chooser dialog and get the file to open
      fileChooserVal = fileChooser.showOpenDialog(this);

      // Check if file was selected
      if (fileChooserVal == JFileChooser.APPROVE_OPTION) {
         readFile = fileChooser.getSelectedFile();
```

```
               readFile = fileChooser.getSelectedFile();

               // Update selected file field
               selectedFileField.setText(readFile.getName());

               // Ensure file is valid
               if (readFile.canRead()) {
                  try {
                     fileByteStream = new FileInputStream(readFile);
                     inFS = new Scanner(fileByteStream);

                     // Clear output area
                     outputArea.setText("");

                     // Read until end-of-file
                     while (inFS.hasNext()) {
                        readLine = inFS.nextLine();
                        outputArea.append(readLine + "\n");
                     }

                  } catch (IOException e) {
                     outputArea.append("\n\nError occurred while creating file stream! " +
                  }
               }
               else { // Can't read file
                  // Show failure dialog
                  JOptionPane.showMessageDialog(this, "Can't read file!");
               }
            }

            return;
         }

         /* Creates a FileReadFrame and makes it visible */
         public static void main(String[] args) {
            // Creates FileReadFrame and its components
            FileReadFrame myFrame = new FileReadFrame();

            myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            myFrame.pack();
            myFrame.setVisible(true);

            return;
         }
      }
```

Screenshot:

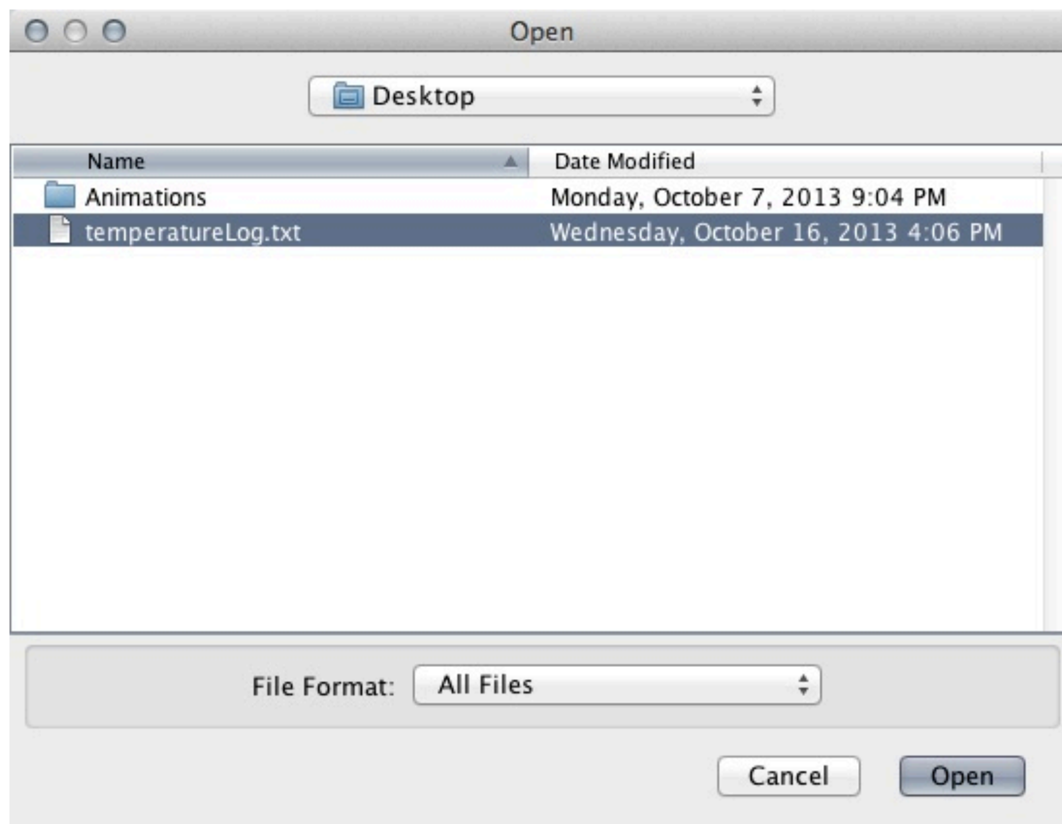


15:58 73.4

The statement `fileChooser = new JFileChooser();` within FileReadFrame's constructor creates a JFileChooser object. By default, a file chooser initially displays files within the user's default directory. The identity of the default directory depends on the operating system in which the program executes (e.g., "My Documents" on Windows, or "/home/<username>" on Linux). The programmer can optionally specify a different initial directory by providing a directory path as a String (or File) argument (e.g. `fileChooser = new JFileChooser(pathString);`). Refer to Oracle's Java JFileChooser class specification for a more comprehensive description of all available constructors.

A GUI program does not typically add a JFileChooser component to a top-level container such as a JFrame. Instead, the program creates a separate dialog window. In the above program, pressing the "Open file" button invokes the actionPerformed() method, which creates a separate dialog window containing the JFileChooser component. The statement `fileChooserVal = fileChooser.showOpenDialog(this);` uses JFileChooser's showOpenDialog() method to create the dialog window containing the file chooser, as shown in the figure below.

Figure 16.11.2: A separate dialog windows containing the JFileChooser component.



The JFileChooser component allows the user to navigate through the file system in order to select a file. A user selects a file by navigating to the containing directory, selecting the file, and pressing the "Open" button. Once the user presses either the "Open" or "Cancel" button, the dialog window closes, restoring control to the parent frame. The showOpenDialog() method returns an integer value denoting the specific operation (i.e., either "Open" or "Cancel") the user selected. The constant APPROVE_OPTION, which is a field within the JFileChooser class, indicates that the user selected a file and pressed the "Open" button. Similarly, the constant CANCEL_OPTION indicates that the user cancelled the operation by pressing the "Cancel" button. A return value of ERROR_OPTION, however, indicates the occurrence of an unanticipated error.

If the user selects a file, the actionPerformed() method then gets a reference to the file's corresponding ***File object*** by calling JFileChooser's getSelectedFile() method and assigning the returned File reference to the variable readFile, using the statement `readFile = fileChooser.getSelectedFile();`. Otherwise, if the user does not select a file, the actionPerformed() method simply returns.

The File class, which the programmer can access by including the import statement `import java.io.File;`, represents a file (or directory path) and defines methods that allow a programmer to ascertain information such as the file's name, location, and access permissions. Refer

to Oracle's Java File class specification for a more comprehensive description of all methods defined within the File class.

If the File is readable (i.e., `readFile.canRead()` returns true), the actionPerformed() method creates a FileInputStream stream and a corresponding Scanner object to read the entire file and print the file's contents to the GUI's text area. Otherwise, if the file is not readable, the program displays a message dialog with the message "Can't read file!".

The statements that create the file stream and read the file are enclosed in a try-catch block in order to detect and handle any exceptions (i.e., errors) encountered while creating the FileInputStream. Exception handling for file I/O is discussed in more detail elsewhere. For now, note that a programmer must use try-catch blocks to enclose statements that create and use a FileInputStream within the actionPerformed() method.

Participation Activity

16.11.1: Terms for file reading with a GUI.

canRead() | JFileChooser | getSelectedFile() | File | CANCEL_OPTION | APPROVE_OPTION

Drag and drop above item	Class for a Swing GUI component that allows a user to select a file.
	Method defined in the File class that indicates if a program can read a file.
	Class that represents a file.
	Constant field defined in the JFileChooser class that indicates the user pressed the "Cancel" button in the JFileChooser.
	Method defined in the JFileChooser class that returns a reference to a File object, which represents the user's selected file.
	Constant field defined in the JFileChooser class that indicates the user selected a file and pressed the "Open" button in the JFileChooser.

Reset

Exploring further:

- How to use various Swing components from Oracle's Java tutorials
- Oracle's Java File class specification