3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes

- Fig. 3.16: Summary of the GUI and Graphics Case Study in each chapter.

Location	Title—Exercise(s)
Section 3.8	Using Dialog Boxes—Basic input and output with dialog boxes
Section 4.14	Creating Simple Drawings—Displaying and drawing lines on the screen
Section 5.10	Drawing Rectangles and Ovals—Using shapes to represent data
Section 6.13	Colors and Filled Shapes—Drawing a bull's-eye and random graphics
Section 7.15	Drawing Arcs—Drawing spirals with arcs
Section 8.16	Using Objects with Graphics—Storing shapes as objects
Section 9.8	Displaying Text and Images Using Labels—Providing status information
Section 10.8	Drawing with Polymorphism—Identifying the similarities between shapes
Exercise 14.17	Expanding the Interface—Using GUI components and event handling
Exercise 15.31	Adding Java 2D—Using the Java 2D API to enhance drawings

Fig. 3.16 | Summary of the GUI and Graphics Case Study in each chapter.

3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes (Cont.)

- Many applications use windows or dialog boxes (also called dialogs) to display output.
- Typically, dialog boxes are windows in which programs display important messages to users.
- Class JOptionPane provides prebuilt dialog boxes that enable programs to display windows containing messages—such windows are called message dialogs.

```
1  // Fig. 3.17: Dialog1.java
2  // Using JOptionPane to display multiple lines in a dialog box.
3  import javax.swing.JOptionPane; // import class JOptionPane
4
5  public class Dialog1
6  {
7     public static void main( String[] args )
8     {
9        // display a dialog with a message
10       JOptionPane.showMessageDialog( null, "Welcome\nto\nJava" );
11    } // end main
12 } // end class Dialog1
```

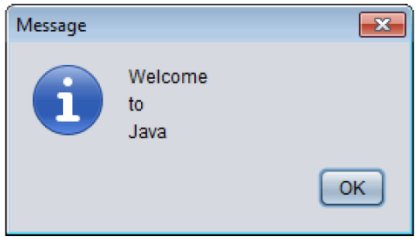

Fig. 3.17 | Using `JOptionPane` to display multiple lines in a dialog box.

3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes (Cont.)

- Package **javax.swing** contains many classes that help you create graphical user interfaces (GUIs).
- GUI components facilitate data entry by a program's user and presentation of outputs to the user.
- `JOptionPane` method showMessageDialog displays a dialog box containing a message.
  - Requires two arguments.
  - The first helps the Java application determine where to position the dialog box.
    - If the first argument is `null`, the dialog box is displayed at the center of your screen.
  - The second argument is the `String` to display in the dialog box.

3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes (Cont.)

- `JOptionPane` method `showMessageDialog` is a static method.
- Such methods often define frequently used tasks.
- Typically called by using method's class name followed by a dot (`.`) and the method name, as in

  ClassName`.`*methodName*`(` *arguments* `)`
- Notice that you do not create an object of class `JOptionPane` to use its `static` method `showMessageDialog`.

3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes (Cont.)

- An **input- dialog** allows the user to enter data into a program.
- `JOptionPane` method showInputDialog displays an input dialog
  - Contains a prompt and a field (known as a text field) in which the user can enter text.
- Method `showInputDialog` (line 11) returns a `String` containing the characters typed by the user.
- If you press the dialog's Cancel button or press the *Esc* key, the method returns `null`.

3.8 (Optional) GUI and Graphics Case Study: Using Dialog Boxes (Cont.)

- `static String` method format returns a formatted `String`.
- Method `format` works like method `System.out.printf`, except that `format` returns the formatted `String` rather than displaying it in a command window.

```
1  // Fig. 3.18: NameDialog.java
2  // Basic input with a dialog box.
3  import javax.swing.JOptionPane;
4
5  public class NameDialog
6  {
7     public static void main( String[] args )
8     {
9        // prompt user to enter name
10       String name =
11          JOptionPane.showInputDialog( "What is your name?" );
12
13       // create the message
14       String message =
15          String.format( "Welcome, %s, to Java Programming!", name );
16
17       // display the message to welcome the user by name
18       JOptionPane.showMessageDialog( null, message );
19    } // end main
20 } // end class NameDialog
```

Displays an input dialog to obtain data from the user

Creates a formatted String containing the name the user entered in the input dialog

Fig. 3.18 | Obtaining user input from a dialog. (Part 1 of 2.)

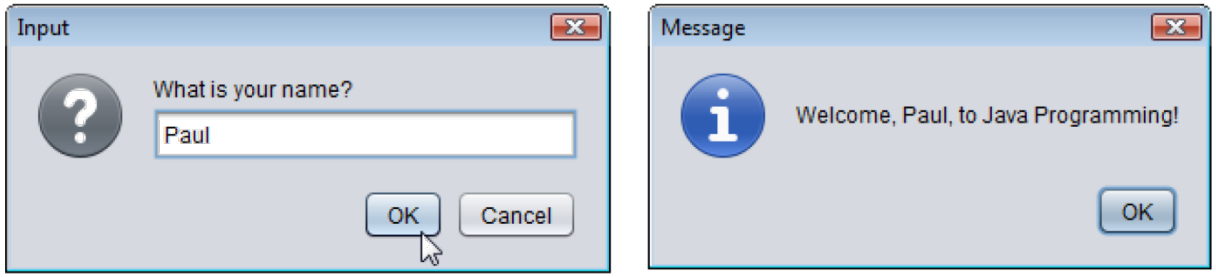

Fig. 3.18 | Obtaining user input from a dialog. (Part 2 of 2.)

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings

- Java's coordinate system is a scheme for identifying points on the screen.
- The upper-left corner of a GUI component has the coordinates (0, 0).
- A coordinate pair is composed of an x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate).
- The *x*-coordinate is the horizontal location (from left to right).
- The *y*-coordinate is the vertical location (from top to bottom).
- The x-axis describes every horizontal coordinate, and the y-axis every vertical coordinate.
- Coordinate units are measured in pixels. The term pixel stands for "picture element." A pixel is a display monitor's smallest unit of resolution.

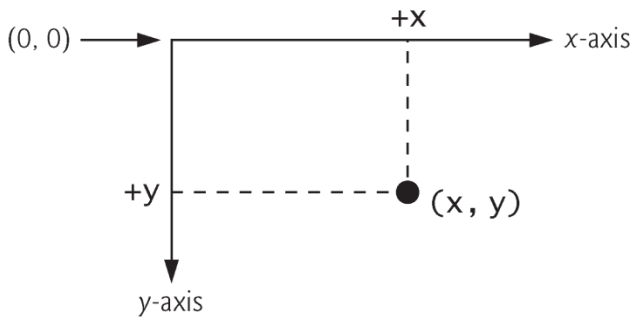

Fig. 4.17 | Java coordinate system. Units are measured in pixels.

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- Class Graphics (from package `java.awt`) provides various methods for drawing text and shapes onto the screen.
- Class JPanel (from package `javax.swing`) provides an area on which we can draw.

```
1  // Fig. 4.18: DrawPanel.java
2  // Using drawLine to connect the corners of a panel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // draws an X from the corners of the panel
9     public void paintComponent( Graphics g )
10    {
11       // call paintComponent to ensure the panel displays correctly
12       super.paintComponent( g );
13
14       int width = getWidth(); // total width
15       int height = getHeight(); // total height
16
17       // draw a line from the upper-left to the lower-right
18       g.drawLine( 0, 0, width, height );
19
20       // draw a line from the lower-left to the upper-right
21       g.drawLine( 0, height, width, 0 );
22    } // end method paintComponent
23 } // end class DrawPanel
```

Import the classes `Graphics` and `JPanel` for use in this source code file.

`DrawPanel` inherits the existing capabilities of class `JPanel`

`paintComponent` *must* be displayed as shown here

This should be the first statement in method `paintComponent`

Determines the width and height of the `DrawPanel` with inherited methods

Draws a line from the top-left to the bottom-right of the `DrawPanel`

Draws a line from the bottom-left to the top-right of the `DrawPanel`

Fig. 4.18 | Using `drawLine` to connect the corners of a panel.

```
1  // Fig. 4.19: DrawPanelTest.java
2  // Application to display a DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main( String[] args )
8     {
9        // create a panel that contains our drawing
10       DrawPanel panel = new DrawPanel();
11
12       // create a new frame to hold the panel
13       JFrame application = new JFrame();
14
15       // set the frame to exit when it is closed
16       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18       application.add( panel ); // add the panel to the frame
19       application.setSize( 250, 250 ); // set the size of the frame
20       application.setVisible( true ); // make the frame visible
21    } // end main
22 } // end class DrawPanelTest
```

Imports class `JFrame` for use in this source code file

Creates a `JFrame` in which the `DrawPanel` will be displayed

Terminate application when window closes

Attach the `DrawPanel` to the `JFrame`

Sets the size of the `JFrame`

Displays the `JFrame` on the screen

Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 1 of 2.)

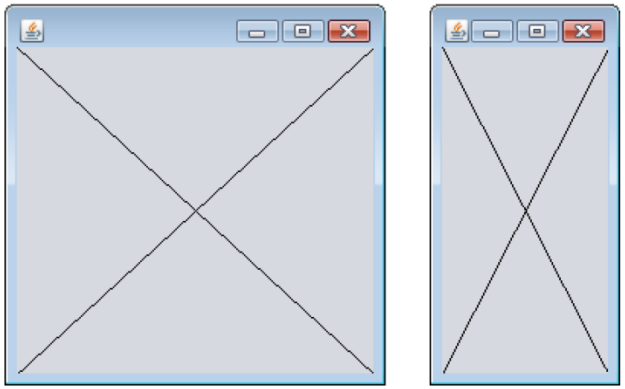

Fig. 4.19 | Creating `JFrame` to display `DrawPanel`. (Part 2 of 2.)

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- The keyword extends creates a so-called inheritance relationship.
- The class from which `DrawPanel` inherits, `JPanel`, appears to the right of keyword `extends`.
- In this inheritance relationship, `JPanel` is called the superclass and `DrawPanel` is called the subclass.

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- `JPanel` has a paintComponent method, which the system calls every time it needs to display the `JPanel`.
- The first statement in every `paintComponent` method you create should always be

```
super.paintComponent( g );
```

- `JPanel` methods getWidth and getHeight return the `JPanel`'s width and height, respectively.
- `Graphics` method drawLine draws a line between two points represented by its four arguments. The first two are the *x*- and *y*-coordinates for one endpoint, and the last two arguments are the coordinates for the other endpoint.

4.14 (Optional) GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- To display the `DrawPanel` on the screen, place it in a window.
- Create a window with an object of class JFrame.
- `JFrame` method setDefaultCloseOperation with the argument JFrame.EXIT_ON_CLOSE indicates that the application should terminate when the user closes the window.
- `JFrame`'s add method attaches the `DrawPanel` (or any other GUI component) to a `JFrame`.
- `JFrame` method setSize takes two parameters that represent the width and height of the `JFrame`, respectively.
- `JFrame` method setVisible with the argument `true` displays the `JFrame`.
- When a `JFrame` is displayed, the `DrawPanel`'s `paintComponent` method is implicitly called

6.13 (Optional) GUI and Graphics Case Study: Colors and Filled Shapes

- Colors displayed on computer screens are defined by their red, green, and blue components (called RGB values) that have integer values from 0 to 255.
- The higher the value of a component color, the richer that shade will be in the final color.
- Java uses class Color in package `java.awt` to represent colors using their RGB values.
- Class `Color` contains 13 predefined `static Color` objects—`BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE` and `YELLOW`.

6.13 (Optional) GUI and Graphics Case Study: Colors and Filled Shapes (Cont.)

- Class `Color` also contains a constructor of the form:
  - `public Color( int r, int g, int b )`
- so you can create custom colors by specifying the red, green and blue component values.
- `Graphics` methods fillRect and fillOval draw filled rectangles and ovals, respectively.
- `Graphics` method setColor sets the current drawing color.

```
1  // Fig. 6.11: DrawSmiley.java
2  // Demonstrates filled shapes.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawSmiley extends JPanel
8  {
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       // draw the face
14       g.setColor( Color.YELLOW );
15       g.fillOval( 10, 10, 200, 200 );
16
17       // draw the eyes
18       g.setColor( Color.BLACK );
19       g.fillOval( 55, 65, 30, 30 );
20       g.fillOval( 135, 65, 30, 30 );
21
22       // draw the mouth
23       g.fillOval( 50, 110, 120, 60 );
24
```

Fig. 6.11 | Drawing a smiley face using colors and filled shapes. (Part 1 of 2.)

```
25          // "touch up" the mouth into a smile
26          g.setColor( Color.YELLOW );
27          g.fillRect( 50, 110, 120, 30 );
28          g.fillOval( 50, 120, 120, 40 );
29       } // end method paintComponent
30    } // end class DrawSmiley
```

Fig. 6.11 | Drawing a smiley face using colors and filled shapes. (Part 2 of 2.)

```
1  // Fig. 6.12: DrawSmileyTest.java
2  // Test application that displays a smiley face.
3  import javax.swing.JFrame;
4
5  public class DrawSmileyTest
6  {
7     public static void main( String[] args )
8     {
9        DrawSmiley panel = new DrawSmiley();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 230, 250 );
15       application.setVisible( true );
16    } // end main
17 } // end class DrawSmileyTest
```

Fig. 6.12 | Creating `JFrame` to display a smiley face. (Part 1 of 2.)

Fig. 6.12 | Creating `JFrame` to display a smiley face. (Part 2 of 2.)

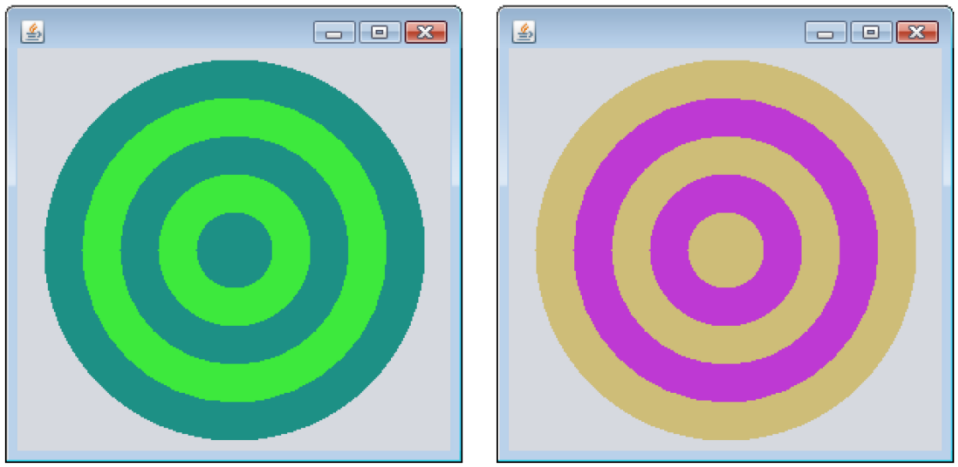

Fig. 6.13 | A bull's-eye with two alternating, random colors.

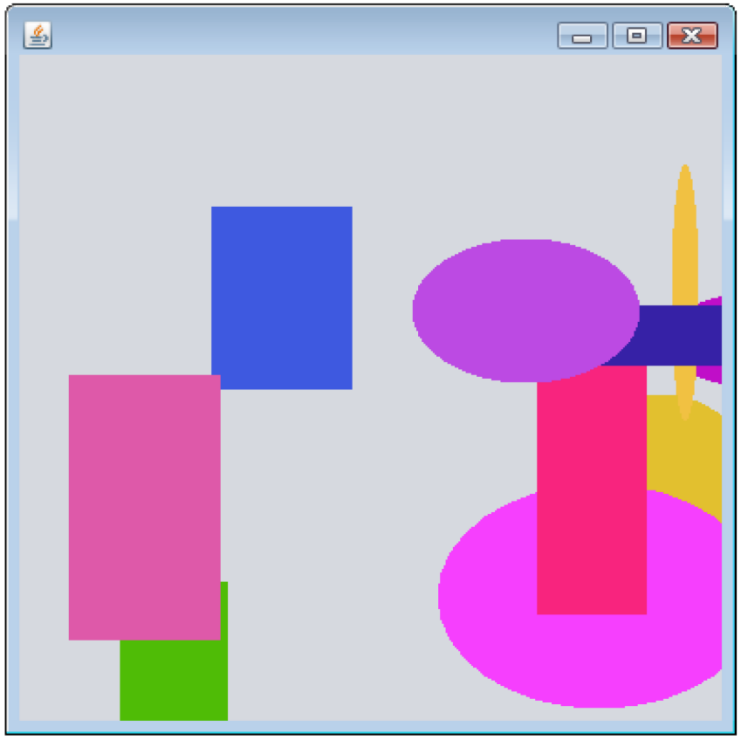

Fig. 6.14 | Randomly generated shapes.