



Objectives

In this tutorial, you will learn to:

- Use simulation techniques that employ random number generation.
- Use methods of class `Random`.
- Generate random numbers.
- Use constants to enhance code readability.
- Use a `JPanel` and a `TitledBorder` to add a border around components.

Outline

- 15.1 Test-Driving the **Craps Game** Application
- 15.2 Random Number Generation
- 15.3 Using Constants in the **Craps Game** Application
- 15.4 Using Random Numbers in the **Craps Game** Application
- 15.5 Wrap-Up

Craps Game Application

Introducing Random Number Generation and the `JPanel`

You will now study a popular type of application involving simulation and game playing. In this tutorial, you will develop a **Craps Game** application. There is something in the air of a gambling casino that invigorates people—from the high rollers at the plush mahogany-and-felt Craps tables to the quarter-poppers at the one-armed bandits. It is the element of chance—the possibility that luck will convert a pocketful of money into a mountain of wealth. Unfortunately, that rarely happens because the odds, of course, favor the casinos.

The element of chance can be introduced into computer applications using random numbers. This tutorial's **Craps Game** application introduces random number generation and the `JPanel` component. It also uses important concepts that you learned earlier in this book, including constants, instance variables, methods and the `switch` multiple-selection statement.

15.1 Test-Driving the Craps Game Application

One of the most popular games of chance is a dice game known as “Craps,” played in casinos throughout the world. This application must meet the following requirements:

Application Requirements

Create an application that simulates playing the world-famous dice game “Craps.” In this game, a player rolls two dice. Each die has six faces. Each face contains 1, 2, 3, 4, 5 or 6 spots. After the dice have come to rest, the sum of the spots on the two top faces is calculated. If the sum is 7 or 11 on the first roll, the player wins. If the sum is 2, 3 or 12 on the first roll (called “craps”), the player loses (the “house” wins). If the sum is 4, 5, 6, 8, 9 or 10 on the first roll, that sum becomes the player’s “point.” To win, a player must continue rolling the dice until the player rolls the point value. The player loses by rolling a 7 before rolling the point.

You begin by test-driving the completed application. Then, you will learn the additional Java technologies you will need to create your own version of this application.

Test-Driving the Craps Game Application

1. **Locating the completed application.** Open the **Command Prompt** window by selecting **Start > Programs > Accessories > Command Prompt**. Change to your completed **Craps Game** application directory by typing `cd C:\Examples\Tutorial15\CompletedApplication\CrapsGame`.
2. **Running the Craps Game application.** Type `java CrapsGame` in the **Command Prompt** window to run the application (Fig. 15.1).

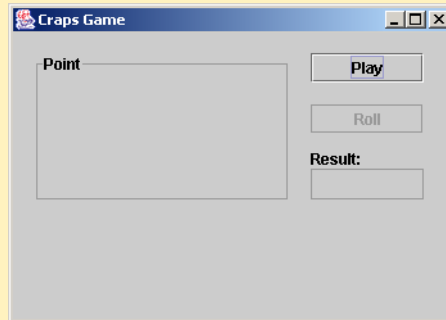


Figure 15.1 Initial appearance of **Craps Game** application.

3. **Starting the game.** Click the **Play** JButton to make the first roll of the dice. There are three possible outcomes at this point. The player wins by rolling a 7 or an 11 (Fig. 15.2). The player loses by rolling a 2, a 3 or a 12 (Fig. 15.3). Otherwise, the roll becomes the player's point (4, 5, 6, 8, 9 or 10), and the dice are displayed as images in JLabels for the remainder of the game (Fig. 15.4). Note that unlike the real game of Craps, the value of the roll is computed in this application using the forward-facing die faces instead of the top faces.



Figure 15.2 Player winning on first roll by rolling 7.



Figure 15.3 Player losing on first roll by rolling 3.

(cont.)



Figure 15.4 Player's first roll setting the point that the player must match to win.

4. **Continuing the game.** If the player does not win or lose on the first roll, the application displays **Roll again!**, as in Fig. 15.4. Click the **Roll** JButton repeatedly until either you win by matching your point value (Fig. 15.5) or you lose by rolling a 7 (Fig. 15.6). When the game ends, you can click the **Play** JButton to start a new game.



Figure 15.5 Player winning the game by matching the point before rolling a 7.



Figure 15.6 Player losing by rolling a 7 before matching the point.

5. **Closing the running application.** Close the running application by clicking its close button.
6. **Closing the Command Prompt window.** Close the **Command Prompt** window by clicking its close button.

15.2 Random Number Generation

Now you will learn how to use an object of class **Random** to introduce the element of chance into your applications. You will learn more about working with objects of the Java class library over the next few tutorials, then you will learn to create your own classes and objects of those classes in Tutorial 18. Consider the following statements:

```
Random randomGenerator = new Random();
int randomNumber = randomGenerator.nextInt();
```

The first statement declares `randomGenerator` as a variable of type `Random` and assigns it a **reference** to a `Random` object. A reference is a variable that refers to an object. A reference specifies the location in the computer's memory of an object. The keyword **new** creates an object and assigns it a location in memory.

The second statement declares `int` variable `randomNumber`. The statement then assigns to `randomNumber` the value returned by calling the `nextInt` method on the `Random` object `randomGenerator`. The **nextInt** method generates a random `int` value selected from all possible `int` values (positive and negative). You can use the `nextInt` method to generate random values of type `int`, or you can use the **nextDouble** method to generate random values of type `double`. The `nextDouble` method returns a positive `double` value between 0.0 and 1.0 (not including 1.0). Class `Random` also contains methods to randomly generate values of the primitive types `boolean`, `float` and `long`.

If the `nextInt` method was to produce truly random values, then every `int` value would have an equal chance (or probability) of being chosen when `nextInt` is called. The `nextInt` method comes close to achieving this goal.

The range of values produced by `nextInt` often is different from the range needed in a particular application. For example, an application that simulates coin tossing might require only the random integers 0 for "heads" and 1 for "tails." An application that simulates the rolling of a six-sided die might require only the random integers from 1 to 6. Similarly, an application that randomly predicts the next type of spaceship (out of four possibilities) that flies across the horizon in a video game might require only the random integers from 1 to 4.

By passing an argument to the `nextInt`¹ method as follows

```
value = 1 + randomGenerator.nextInt( 6 );
```

you can produce random integers in the range from 1 to 6. When a single argument is passed to `nextInt`, the values returned by `nextInt` will be in the range from 0 to one less than the value of that argument (5 in the preceding statement). You can change the range of numbers produced by `nextInt` by adding 1 to the previous result, so that the return values are between 1 and 6, rather than 0 and 5. That new range, 1 to 6, corresponds nicely with the roll of a six-sided die, for example.

As with the `nextInt` method, the range of values produced by the `nextDouble` method (that is, values greater than or equal to 0.0 and less than 1.0) is also usually different from the range needed in a particular application. By multiplying the value returned from the `nextDouble` method as follows

```
doubleValue = 10 * randomGenerator.nextDouble();
```

you can produce `double` values in the range from 0.0 to 10.0 (not including 10.0). Figure 15.7 shows examples of the ranges of random numbers returned by expressions containing calls to methods `nextInt` and `nextDouble`.

Expression	Resulting range
<code>randomGenerator.nextInt()</code>	(-2^{32}) to $(2^{32} - 1)$ [all possible values of <code>int</code>]
<code>randomGenerator.nextInt(30)</code>	0 to 29

Figure 15.7 `nextInt` and `nextDouble` method call expressions with ranges of random numbers produced. (Part 1 of 2.)

1. In Tutorial 12, you learned that the number, type and order of arguments to a method must exactly match the method declaration. You may have noticed that the `nextInt` method of class `Random` can be called using zero arguments or one argument. Java has a capability called **method overloading** that allows several methods to have the same name but different numbers or types of arguments.

Expression	Resulting range
<code>10 + randomGenerator.nextInt(10)</code>	10 to 19
<code>randomGenerator.nextDouble()</code>	0.0 to less than 1.0
<code>8 * randomGenerator.nextDouble()</code>	0.0 to less than 8.0

Figure 15.7 `nextInt` and `nextDouble` method call expressions with ranges of random numbers produced. (Part 2 of 2.)

SELF-REVIEW

- The statement _____ returns an integer in the range 8–300.
 - `7 + randomObject.nextInt(293);`
 - `8 + randomObject.nextInt(292);`
 - `8 + randomObject.nextInt(293);`
 - None of the above.
- The statement _____ returns a number in the range 15–35.
 - `10 + randomObject.nextInt(26);`
 - `15 + randomObject.nextInt(21);`
 - `10 + randomObject.nextInt(25);`
 - `15 + randomObject.nextInt(35);`

Answers: 1) c. 2) b.

15.3 Using Constants in the Craps Game Application

The following pseudocode describes the basic operation of the **Craps Game** application when the **Play** JButton is clicked:

When the player clicks the Play JButton

Roll the two dice using random numbers

Calculate the sum of the two dice

Display images of the rolled dice

Switch based on the sum of the two dice:

Case where the sum is 7 or 11

Display the winning message

Case where the sum is 2, 3 or 12

Display the losing message

Default case

Set the value of the point to the sum of the dice and display the value

Disable the Play JButton and enable the Roll JButton

When the player clicks the Roll JButton

Roll the two dice using random numbers

Calculate the sum of the two dice

Display images of the rolled dice

If the player rolls the point

Display the winning message

Clear the value of the point

Enable the Play JButton and disable the Roll JButton

If the player rolls a 7

Display the losing message

Clear the value of the point

Enable the Play JButton and disable the Roll JButton

Now that you have test-driven the **Craps Game** application and studied its pseudocode representation, you will use an ACE table to help you convert the pseudocode to Java. Figure 15.8 lists the actions, components and events that will help you complete your own version of this application.

Action/Component/ Event (ACE) Table for the Craps Game Application

Action	Component	Event
Label the application's components	resultJLabel	
	playJButton	User clicks Play JButton
Roll the two dice using random numbers	randomObject (Random)	
Calculate the sum of the two dice		
Display images of the rolled dice	die1JLabel, die2JLabel	
Switch based on the sum of the two dice:		
Case where the sum is 7 or 11 Display the winning message	resultJTextField	
Case where the sum is 2, 3 or 12 Display the losing message	resultJTextField	
Default case Display the value of the point	pointDie1JLabel, pointDie2JLabel	
Disable the Play JButton and enable the Roll JButton	playJButton, rollJButton	
	rollJButton	User clicks Roll JButton
Roll the two dice using random numbers	randomObject (Random)	
Calculate the sum of the two dice		
Display images of the rolled dice	die1JLabel, die2JLabel	
If the player rolls the point Display the winning message	resultJTextField	
Clear the value of the point	pointDie1JLabel, pointDie2JLabel	
Enable the Play JButton and disable the Roll JButton	playJButton, rollJButton	
If the player rolls a 7 Display the losing message	resultJTextField	
Clear the value of the point	pointDie1JLabel, pointDie2JLabel	
Enable the Play JButton and disable the Roll JButton	playJButton, rollJButton	

Figure 15.8 ACE table for the **Craps Game** application.

In the following boxes, you will create an entertaining application that simulates playing the game Craps. As you have learned, Java has access to the Java class library, which is a rich collection of classes that can be used to enhance applications. The Java class library includes classes for enabling your applications to use files, graphics, multimedia, perform arithmetic operations and much more. These pre-defined classes are grouped into categories of related classes called **packages**. The **java.util** package provides random number processing capabilities with class **Random**. Importing a class (using the keyword **import**) allows your application to access that class. You will need to use code to generate random numbers for the **Craps Game** application; therefore, you will now import the **Random** class from the **java.util** package.

Importing the Random Class from the Java Class Library

1. **Copying the template to your working directory.** Copy the C:\Examples\Tutorial15\TemplateApplication\CrapsGame directory to your C:\SimplyJava directory.
2. **Opening the Craps Game application's template file.** Open the template file CrapsGame.java in your text editor.
3. **Importing class Random.** Add line 5 of Fig. 15.9 into your code. This line imports class Random from the java.util package. Lines 3, 4, 6 and 7 are also import declarations. These lines use an asterisk (*), which allows the application to access entire packages. These packages allow you to create and manipulate GUIs, and enable event handling for GUI components.

Importing the java.util.Random class

```

Source Editor [CrapsGame *.java]
1 // Tutorial 15: CrapsGame.java
2 // This application plays a simple craps game
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.util.Random;
6 import javax.swing.*;
7 import javax.swing.border.*;
  
```

Figure 15.9 Importing class Random of the java.util package.

4. **Saving the application.** Save your modified source code file.

Notice that the numbers 2, 3, 7, 11 and 12 have special meanings during a game of Craps. It would be helpful to create these constants and assign them meaningful names for use in your application. Java allows you to create a constant with the keyword final. You will now create constants whose identifiers describe significant dice combinations in Craps (such as SNAKE_EYES, TREY, CRAPS, LUCKY_SEVEN, YO_LEVEN and BOX_CARS). You will use these to enhance the readability of your code and ensure that numbers are consistent throughout your application.

Declaring Constants and Instance Variables

1. **Declaring constants.** Add lines 31–39 of Fig. 15.10 to your application. Recall that constant declarations contain the keyword final before the type of the variable. Notice that you can assign the same value to multiple constants, as in lines 32 and 39—in this case, because 7 has a different meaning on the first roll than on subsequent rolls.

Declaring constants

```

Source Editor [CrapsGame *.java]
29 private JTextField resultJTextField;
30
31 // constants representing winning dice rolls
32 private final int LUCKY_SEVEN = 7;
33 private final int YO_LEVEN = 11;
34
35 // constants representing losing dice rolls
36 private final int SNAKE_EYES = 2;
37 private final int TREY = 3;
38 private final int BOX_CARS = 12;
39 private final int CRAPS = 7;
40
41 // no-argument constructor
  
```

Figure 15.10 Declaring constants in the Craps Game application.

- (cont.)
2. Add lines 41–43 of Fig. 15.11 to your application. In this application, you will need to access images that display the six faces of a die. For convenience, each file has a name that differs only by one number. For example, the image for the die face displaying 1 is named `die1.png`, and the image for the die face displaying 6 is named `die6.png`. Recall that `png` is an image file name extension that is short for Portable Network Graphics. These images are stored in the directory named `Images` in your working directory, `C:\SimplyJava\CrapsGame`. As such, the `String` `Images/die1.png` would correctly indicate the location of the die face image displaying 1 relative to your working directory. To help create a `String` representing the path to the image, `Strings` `FILE_PREFIX` (`Images/die`) and `FILE_SUFFIX` (`.png`) are used (as constants) to store the prefix and suffix of the file name. An image name is thus the combination of `FILE_PREFIX`, number and `FILE_SUFFIX`.

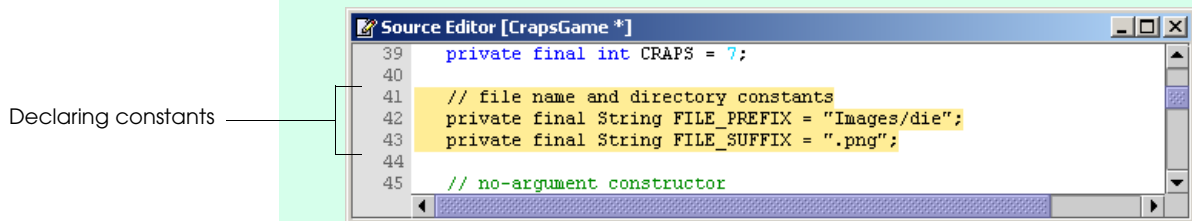


Figure 15.11 Declaring constants in the **Craps Game** application.

3. **Declaring instance variables.** Add lines 45–47 of Fig. 15.12 below the constant declarations to declare and initialize two instance variables. The game of Craps requires that you store the user's point, once established on the first roll, for the duration of the game. Therefore, variable `myPoint` (line 46) is declared as an `int` to store the sum of the dice on the first roll. You will use the `Random` object referenced by `randomObject` (line 47) to “roll” the dice.

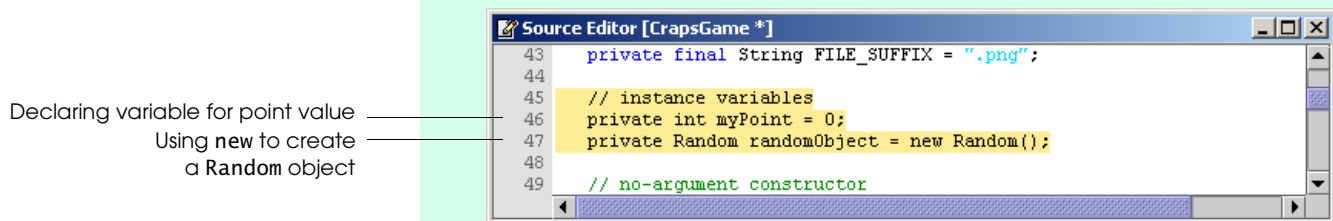


Figure 15.12 Adding instance variables to the **Craps Game** application.

4. **Saving the application.** Save your modified source code file.

You have now declared constants and instance variables for your application. Before you use these variables, you must first customize a **JPanel**. A **JPanel** is a container (like the content pane you learned about in [***XREF:2***]) that allows you to group related components. Multiple **JPanel**s can be added to your application's content pane. In this application, you use a **JPanel** to add a **TitledBorder** around two **JLabels**. A **TitledBorder** places a line and a title around a GUI component. Any GUI component attached to the component with the border will appear inside the border. A **border** can be added to any GUI component by setting its **border** property. Some GUI components (such as **JButtons** and **JTextFields**) already have default borders.

Adding a TitledBorder

1. **Customizing a JPanel.** Add lines 69–70 of Fig. 15.13. Line 69 sets the *bounds* property of the JPanel. Line 70 sets the JPanel’s *layout* property to null using the `setLayout` method. The *layout* property controls how components are arranged on a JPanel. Setting the value to null allows **absolute positioning** of components. Absolute positioning means that you specify exactly where the component will appear on a container (such as a JPanel). Other layouts usually involve **relative positioning**, which means that components are placed in relation to other components in a container. Line 71 adds the JPanel to the content pane. This JPanel will contain the two JLabels that display the images of the point dice (`pointDie1JLabel` and `pointDie2JLabel`).

Customizing the JPanel

```

65   pointDiceTitledBorder = new TitledBorder( "Point" );
66
67   // set up pointDiceJPanel
68   pointDiceJPanel = new JPanel();
69   pointDiceJPanel.setBounds( 16, 16, 200, 116 );
70   pointDiceJPanel.setLayout( null );
71   contentPane.add( pointDiceJPanel );
72
73   // set up pointDie1JLabel
    
```

Figure 15.13 Customizing pointDiceJPanel.

2. **Adding components to the JPanel.** Add lines 75–76 and lines 80–81 of Fig. 15.14. Line 75 sets the *bounds* property of `pointDie1JLabel`. Line 76 adds `pointDie1JLabel` to `pointDiceJPanel`. Recall that the first two values of the *bounds* property are the *x* and *y* values of the component. Because you add `pointDie1JLabel` to `pointDiceJPanel`, not to the content pane, *x* and *y* values of 0, 0 refer to the upper left corner of `pointDiceJPanel`, not of the content pane. In this case, line 69 of Fig. 15.13 specifies that the upper-left corner of the JPanel is located at 16, 16 on the content pane. Lines 80–81 set the *bounds* property of `pointDie2JLabel` and add it to `pointDiceJPanel`.

Adding a JLabel to the JPanel

Adding a JLabel to the JPanel

```

71   contentPane.add( pointDiceJPanel );
72
73   // set up pointDie1JLabel
74   pointDie1JLabel = new JLabel();
75   pointDie1JLabel.setBounds( 24, 34, 64, 56 );
76   pointDiceJPanel.add( pointDie1JLabel );
77
78   // set up pointDie2JLabel
79   pointDie2JLabel = new JLabel();
80   pointDie2JLabel.setBounds( 120, 34, 64, 56 );
81   pointDiceJPanel.add( pointDie2JLabel );
82
83   // set up die1JLabel
    
```

Figure 15.14 Adding components to the JPanel.

3. **Setting the border of the JPanel.** Look at line 65 of Fig. 15.15 which creates a `TitledBorder` with the title "Point". This line uses the keyword `new` to create a `TitledBorder` object. Add line 71 of Fig. 15.15. This line calls the `setBorder` method to set the *border* property of `pointDiceJPanel`. The `TitledBorder` object, `pointDiceTitledBorder`, is passed to the `setBorder` method to add `pointDiceTitledBorder` to `pointDiceJPanel`. This border shows up as a thin line around `pointDiceJPanel`. The title is displayed in the upper-left part of the border.

(cont.)

Creating a TitledBorder

Setting the *border* property

```

64 // set up pointDiceTitledBorder for use with pointDiceJPanel
65 pointDiceTitledBorder = new TitledBorder( "Point" );
66
67 // set up pointDiceJPanel
68 pointDiceJPanel = new JPanel();
69 pointDiceJPanel.setBounds( 16, 16, 200, 116 );
70 pointDiceJPanel.setLayout( null );
71 pointDiceJPanel.setBorder( pointDiceTitledBorder );
72 contentPane.add( pointDiceJPanel );

```

Figure 15.15 Setting the border of pointDiceJPanel.

4. **Saving the application.** Save your modified source code file.
5. **Opening the Command Prompt window and changing directories.** Open the Command Prompt window by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\CrapsGame`.
6. **Compiling the application.** Compile your application by typing `javac CrapsGame.java`.
7. **Running the application.** When your application compiles correctly, run it by typing `java CrapsGame`. Figure 15.16 shows the updated application running. You have not coded methods for the **Play** or **Roll** JButtons yet, so your application will not function. You will code these methods shortly.

TitledBorder is displayed with Point as the title

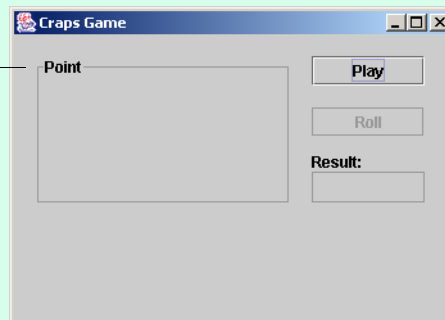


Figure 15.16 Running Craps Game application.

8. **Closing the application.** Close your running application by clicking its close button.
9. **Closing the Command Prompt window.** Close the Command Prompt window by clicking its close button.

SELF-REVIEW

1. Use the keyword _____ to define constants.
 - a) readOnly
 - b) final
 - c) constants
 - d) constant
2. Package _____ contains class Random.
 - a) java.awt
 - b) java.utility
 - c) java.swing
 - d) java.util

Answers: 1) b. 2) d.

15.4 Using Random Numbers in the Craps Game Application

Now you will add code that executes when the user clicks the **Craps Game** application's JButtons. You begin by inserting the code that will execute when the user clicks the **Play** JButton.

Coding the playJButtonActionPerformed Method

1. **Removing Images from a JLabel and rolling dice.** Begin coding the playJButtonActionPerformed method by adding lines 156–164 of Fig. 15.17. Lines 157–158 remove any images from the JLabels used to display the “point dice.” Though there are no images when the application is first run, the images from the previous game must be cleared if the user chooses to play again. Setting the *icon* property to the keyword `null` indicates that there is no image to display.

Removing images from JLabels

Setting the title of the border and updating the JPanel

“Rolling” the dice

```

Source Editor [CrapsGame *]
153 // start new game of craps
154 private void playJButtonActionPerformed( ActionEvent event )
155 {
156 // clear point icons
157 pointDie1JLabel.setIcon( null );
158 pointDie2JLabel.setIcon( null );
159
160 // reset title of border
161 pointDiceTitledBorder.setTitle( "Point" );
162 pointDiceJPanel.repaint();
163
164 int sumOfDice = rollDice(); // roll dice
165
166 } // end method playJButtonActionPerformed
    
```

Figure 15.17 Clearing images and rolling the dice.

Line 161 changes the text displayed on `pointDiceTitledBorder` to "Point", using the `setTitle` method. This method sets the `TitledBorder`'s *title* property, which controls the text that is displayed in the border. The `rollJButtonActionPerformed` method (declared later) changes the title of the border. This line resets the title of the border. Line 162 redraws the `JPanel` using the `repaint` method. The `repaint` method is used when a component needs to be updated, such as when the title of its border is changed. Once the `repaint` method is called, the `JPanel` is redrawn as soon as possible so that the title of the border is displayed with the current value (set at line 161).

Line 164 declares an `int` variable `sumOfDice` and assigns to it the value returned by rolling the dice. This is accomplished by calling the `rollDice` method, which you will define later in this tutorial. The `rollDice` method rolls the two dice, displays the dice images in the lower two `JLabels` and returns the sum of the dice values.

2. **Using a switch statement to determine the result of rolling the dice.** Recall that if the player rolls 7 or 11 on the first roll, the player wins, and if the player rolls 2, 3 or 12 on the first roll, the player loses. To enable your application to handle the cases in which the player wins or loses on the first roll, add lines 166–182 of Fig. 15.18 to the `playJButtonActionPerformed` method beneath the code you added in the previous step.

(cont.)

Winning on the first roll

Losing on the first roll

```

164     int sumOfDice = rollDice(); // roll dice
165
166     // check results of the first dice roll
167     switch ( sumOfDice )
168     {
169         // win on first roll
170         case LUCKY_SEVEN:
171         case YO_LEVEN:
172             resultJTextField.setText( "You win!!!" );
173             break;
174
175         // lose on first roll
176         case SNAKE_EYES:
177         case TREY:
178         case BOX_CARS:
179             resultJTextField.setText( "Sorry, you lose." );
180             break;
181
182     } // end switch statement
183

```

Figure 15.18 switch statement in playJButtonActionPerformed.

The first case (lines 170–173) executes for first-roll values of 7 or 11, using the constant values `LUCKY_SEVEN` and `YO_LEVEN`. Recall that several case labels can be specified to execute the same statements. If the sum of the dice is 7 (`LUCKY_SEVEN`) or 11 (`YO_LEVEN`), the code at line 172 displays "You win!!!" in `resultJTextField`. If the sum of the dice is 2 (`SNAKE_EYES`), 3 (`TREY`) or 12 (`BOX_CARS`), the code at line 179 executes and displays "Sorry, you lose." in `resultJTextField`. Line 182 ends the switch statement.

3. **Using the default case to continue the game.** Add lines 182–200 of Fig. 15.19. If the player did not roll a 2, 3, 7, 11 or 12, then the sum of the dice becomes the point and the player must roll again. Line 186 in the default case's body sets instance variable `myPoint` to the sum of the die values. Next, line 187 sets the `text` property of `resultJTextField` to notify the user to roll again.

Player must match the point

Displaying the die images

Displaying the point and updating the application

Allowing the player to roll again

```

180         break;
181
182         // remember point in instance variable
183         default:
184
185             // set the point and output result
186             myPoint = sumOfDice;
187             resultJTextField.setText( "Roll again!" );
188
189             // show the dice images
190             pointDie1JLabel.setIcon( die1JLabel.getIcon() );
191             pointDie2JLabel.setIcon( die2JLabel.getIcon() );
192
193             // update the border title
194             pointDiceTitledBorder.setTitle(
195                 "Point is " + sumOfDice );
196             pointDiceJPanel.repaint();
197
198             // change the state of the JButtons
199             playJButton.setEnabled( false );
200             rollJButton.setEnabled( true );
201
202     } // end switch statement

```

Figure 15.19 default case in playJButtonActionPerformed.

(cont.)

The user must match the point to win, so you will display the images corresponding to the point roll. This is done by setting the images in the point JLabels to the images currently in the die JLabels. In Tutorial 2, you learned how to display an image in a JLabel by setting its *icon* property. Lines 190–191 display the die faces for the point JLabels by setting the *icon* property of each point JLabel to the value of the *icon* property of its corresponding die JLabel (retrieved using the *getIcon* method of JLabel). Lines 194–195 change the title on the pointDiceTitledBorder, using its *title* property to display the value of the current point. Finally, the **Play** JButton is disabled (line 199) and the **Roll** JButton is enabled (line 200), limiting users to clicking the **Roll** JButton for all the rolls in the rest of the game.

4. **Saving the application.** Save your modified source code file.

The **Roll** JButton is enabled after the user has started the game. The user presses the **Roll** JButton and tries to match the point. Next, you code the event handler method for the **Roll** JButton.

Coding the RollJButtonActionPerformed Method

1. **Rolling the dice.** The user clicks the **Roll** JButton to roll the dice and try to match the point. Begin coding the `rollJButtonActionPerformed` method by adding line 209 of Fig. 15.20. Line 209 declares an `int` variable `sumOfDice`, calls the `rollDice` method to roll the dice and display the die images, and assigns the sum of the dice to variable `sumOfDice`.

"Rolling" the dice

```

Source Editor [CrapsGame *]
206 // continue the game
207 private void rollJButtonActionPerformed( ActionEvent event )
208 {
209     int sumOfDice = rollDice(); // roll dice
210
211 } // end method rollJButtonActionPerformed
  
```

Figure 15.20 Rolling the dice in `rollJButtonActionPerformed`.

2. **Determining the output of the roll.** If the roll matches the point, the user wins and the game ends. However, if the user rolls a 7 (CRAPS), the user loses and the game ends. Add lines 211–224 of Fig. 15.21 in the `rollJButtonActionPerformed` method.

Displaying winning message

Displaying losing message

```

Source Editor [CrapsGame *]
209     int sumOfDice = rollDice(); // roll dice
210
211     // determine outcome of roll, player matches point
212     if ( sumOfDice == myPoint )
213     {
214         resultJTextField.setText( "You win!!!" );
215         rollJButton.setEnabled( false );
216         playJButton.setEnabled( true );
217     }
218     // determine outcome of roll, player loses
219     else if ( sumOfDice == CRAPS )
220     {
221         resultJTextField.setText( "Sorry, you lose." );
222         rollJButton.setEnabled( false );
223         playJButton.setEnabled( true );
224     }
225 } // end method rollJButtonActionPerformed
  
```

Figure 15.21 Determining the outcome of a roll.

(cont.)

The `if` (line 212) determines whether the sum of the dice in the current roll matches the point. If the sum and point match, the application displays the winning message in `resultJTextField`. It then allows the user to start a new game, by disabling the **Roll** `JButton` and enabling the **Play** `JButton`.

The `else` (line 219) contains an `if` (lines 219–224) that determines whether the sum of the dice in the current roll is 7 (CRAPS). If so, the application displays the message that the user has lost (in `resultJTextField`) and ends the game by disabling the **Roll** `JButton` and enabling the **Play** `JButton`. If the player neither matches the point nor rolls a 7, then the player is allowed to roll again. The player can roll the dice again by clicking the **Roll** `JButton`.

3. **Saving the application.** Save your modified source code file.

Next, you will add code that will simulate rolling the dice and display the dice images in the appropriate `JLabels`.

Using Random Numbers to Simulate Rolling Dice

1. **Using the `Random` object to simulate dice rolling.** At several places in this application, it will be necessary to roll and display two dice. Therefore, it is a good idea to create two methods: one to roll the dice (`rollDice`) and one to display a die image (`displayDie`) that can be called from different locations in the application. Declare the `rollDice` method first, by adding lines 228–241 of Fig. 15.22 after the `rollJButtonActionPerformed` method.

Getting two random numbers

Displaying the die images

Returning the sum of the dice

```

226 } // end method rollJButtonActionPerformed
227
228 // generate random die rolls
229 private int rollDice()
230 {
231 // generate random die values
232 int die1 = 1 + randomObject.nextInt( 6 );
233 int die2 = 1 + randomObject.nextInt( 6 );
234
235 // display the dice images
236 displayDie( die1JLabel, die1 );
237 displayDie( die2JLabel, die2 );
238
239 return die1 + die2; // return sum of dice values
240
241 } // end method rollDice
242
243 // main method

```

Figure 15.22 Declaring the `rollDice` method.

This code sets the values of `die1` and `die2` to random integers between 1 to 6, inclusive. Lines 232–233 accomplish this using the expression `1 + randomObject.nextInt(6)`. Remember that the value returned from `nextInt` is a non-negative integer between zero and one less than the argument (in this case, the range 0–5).

The method then makes two calls to `displayDie` (lines 236–237), a method that displays the image of the die corresponding to a face value in the range 1–6. The first parameter in `displayDie` is the `JLabel` that will display the image, and the second parameter is the number that appears on the face of the die. You will declare the `displayDie` method in *Step 2*. Finally, the method returns the sum of the values of the dice (line 239), which the application uses to determine the outcome of the Craps game.

- (cont.)
2. **Displaying the dice images.** You will now declare the `displayDie` method to display the die images corresponding to the random die values generated in the `rollDice` method. Add lines 243–252 of Fig. 15.23 (after the `rollDice` method) to declare the `displayDie` method. This method takes a `JLabel` and an `int` as the arguments. The method uses the `int` to get an image, which is then displayed in the `JLabel`.

Creating a new ImageIcon

Displaying a die image

```

Source Editor [CrapsGame *]
241 } // end method rollDice
242
243 // displays the die image
244 private void displayDie( JLabel picDieJLabel, int face )
245 {
246     ImageIcon image =
247         new ImageIcon( FILE_PREFIX + face + FILE_SUFFIX );
248
249     // display die images in picDieJLabel
250     picDieJLabel.setIcon( image );
251
252 } // end method displayDie
253
254 // main method
    
```

Figure 15.23 Declaring the `displayDie` method.

Lines 246–247 create an `ImageIcon`. Recall that `FILE_PREFIX` and `FILE_SUFFIX` were declared earlier as constants. The String `FILE_PREFIX + face + FILE_SUFFIX` specifies the location of the file (line 247). If the value of `face` is 1, the expression would result in `"Images/die1.png"`. This is the location of the image of a die face showing 1. Line 250 sets the `icon` property for the specified `JLabel` to the `ImageIcon` created in lines 246–247.

3. **Saving the application.** Save your modified source code file.
4. **Opening the Command Prompt window and changing directories.** Open the **Command Prompt** window by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\CrapsGame`.
5. **Compiling the application.** Compile your application by typing `javac CrapsGame.java`.
6. **Running the application.** When your application compiles correctly, run it by typing `java CrapsGame`. Figure 15.24 shows the completed application running.

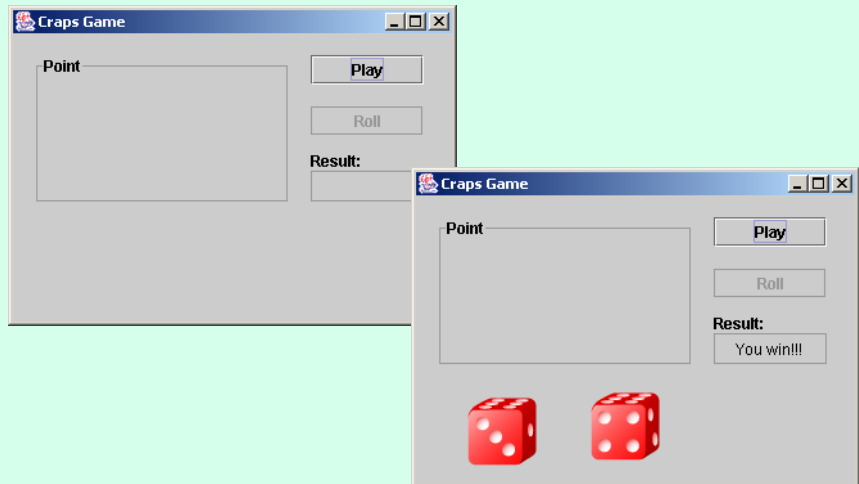


Figure 15.24 Running the completed **Craps Game** application.

- (cont.)
7. **Closing the running application.** Close your running application by clicking its close button.
 8. **Closing the Command Prompt window.** Close the Command Prompt window by clicking its close button.

Figure 15.25 presents the source code for the **Craps Game** application. The lines of code that you added, viewed or modified in this tutorial are highlighted.

```

1 // Tutorial 15: CrapsGame.java
2 // This application plays a simple craps game.
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.util.Random;
6 import javax.swing.*;
7 import javax.swing.border.*;
8
9 public class CrapsGame extends JFrame
10 {
11     // JPanel and TitledBorder to contain dice
12     private JPanel pointDiceJPanel;
13     private TitledBorder pointDiceTitledBorder;
14
15     // JLabels to display the die images in pointDiceJPanel
16     private JLabel pointDie1JLabel;
17     private JLabel pointDie2JLabel;
18
19     // JLabels to display the die images from the rolls of the dice
20     private JLabel die1JLabel;
21     private JLabel die2JLabel;
22
23     // JButtons to allow user to interact with game
24     private JButton playJButton;
25     private JButton rollJButton;
26
27     // JLabel and JTextField show results of game
28     private JLabel resultJLabel;
29     private JTextField resultJTextField;
30
31     // constants representing winning dice rolls
32     private final int LUCKY_SEVEN = 7;
33     private final int YO_LEVEN = 11;
34
35     // constants representing losing dice rolls
36     private final int SNAKE_EYES = 2;
37     private final int TREY = 3;
38     private final int BOX_CARS = 12;
39     private final int CRAPS = 7;
40
41     // file name and directory constants
42     private final String FILE_PREFIX = "Images/die";
43     private final String FILE_SUFFIX = ".png";
44
45     // instance variables
46     private int myPoint = 0;
47     private Random randomObject = new Random();
48

```

Importing the class java.util.Random — 5

Declaring the constants for the dice rolls — 32-39

Declaring the constants for the file name — 42-43

Declaring instance variable myPoint — 46

Creating a Random object using the new keyword — 47

Figure 15.25 Craps Game application code listing. (Part 1 of 5.)

```

49 // no-argument constructor
50 public CrapsGame()
51 {
52     createUserInterface();
53 }
54
55 // create and position GUI components; register event handlers
56 private void createUserInterface()
57 {
58     // get content pane for attaching GUI components
59     Container contentPane = getContentPane();
60
61     // enable explicit positioning of GUI components
62     contentPane.setLayout( null );
63
64     // set up pointDiceTitledBorder for use with pointDiceJPanel
65     pointDiceTitledBorder = new TitledBorder( "Point" );
66
67     // set up pointDiceJPanel
68     pointDiceJPanel = new JPanel();
69     pointDiceJPanel.setBounds( 16, 16, 200, 116 );
70     pointDiceJPanel.setLayout( null );
71     pointDiceJPanel.setBorder( pointDiceTitledBorder );
72     contentPane.add( pointDiceJPanel );
73
74     // set up pointDie1JLabel
75     pointDie1JLabel = new JLabel();
76     pointDie1JLabel.setBounds( 24, 34, 64, 56 );
77     pointDiceJPanel.add( pointDie1JLabel );
78
79     // set up pointDie2JLabel
80     pointDie2JLabel = new JLabel();
81     pointDie2JLabel.setBounds( 120, 34, 64, 56 );
82     pointDiceJPanel.add( pointDie2JLabel );
83
84     // set up die1JLabel
85     die1JLabel = new JLabel();
86     die1JLabel.setBounds( 40, 150, 64, 64 );
87     contentPane.add( die1JLabel );
88
89     // set up die2JLabel
90     die2JLabel = new JLabel();
91     die2JLabel.setBounds( 136, 150, 64, 56 );
92     contentPane.add( die2JLabel );
93
94     // set up playJButton
95     playJButton = new JButton();
96     playJButton.setBounds( 232, 16, 88, 23 );
97     playJButton.setText( "Play" );
98     contentPane.add( playJButton );
99     playJButton.addActionListener(
100
101         new ActionListener() // anonymous inner class
102         {
103             // event handler called when playJButton is clicked
104             public void actionPerformed ( ActionEvent event )
105             {
106                 playJButtonActionPerformed( event );

```

Creating a TitledBorder object — 65

Adding a border to the JPanel — 71

Adding a JLabel to a JPanel using the add method — 77

Adding a JLabel to a JPanel using the add method — 82

Figure 15.25 Craps Game application code listing. (Part 2 of 5.)

```

107         }
108     }
109     } // end anonymous inner class
110
111 ); // end call to addActionListener
112
113 // set up rollJButton
114 rollJButton = new JButton();
115 rollJButton.setBounds( 232, 56, 88, 23 );
116 rollJButton.setText( "Roll" );
117 rollJButton.setEnabled( false );
118 contentPane.add( rollJButton );
119 rollJButton.addActionListener(
120
121     new ActionListener() // anonymous inner class
122     {
123         // event handler called when rollJButton is clicked
124         public void actionPerformed ( ActionEvent event )
125         {
126             rollJButtonActionPerformed( event );
127         }
128     }
129     } // end anonymous inner class
130
131 ); // end call to addActionListener
132
133 // set up resultJLabel
134 resultJLabel = new JLabel();
135 resultJLabel.setBounds( 232, 90, 48, 16 );
136 resultJLabel.setText( "Result:" );
137 contentPane.add( resultJLabel );
138
139 // set up resultJTextField
140 resultJTextField = new JTextField();
141 resultJTextField.setBounds( 232, 106, 88, 24 );
142 resultJTextField.setHorizontalAlignment( JTextField.CENTER );
143 resultJTextField.setEditable( false );
144 contentPane.add( resultJTextField );
145
146 // set properties of application's window
147 setTitle( "Craps Game" ); // set title bar string
148 setSize( 350, 250 ); // set window size
149 setVisible( true ); // display window
150
151 } // end method createUserInterface
152
153 // start new game of craps
154 private void playJButtonActionPerformed( ActionEvent event )
155 {
156     // clear point icons
157     pointDie1JLabel.setIcon( null );
158     pointDie2JLabel.setIcon( null );
159
160     // reset title of border
161     pointDiceTitledBorder.setTitle( "Point" );
162     pointDiceJPanel.repaint();
163
164     int sumOfDice = rollDice(); // roll dice

```

Removing images from JLabels

Setting the title of the border
and updating the JPanel

Figure 15.25 Craps Game application code listing. (Part 3 of 5.)

```

165
166 // check results of the first dice roll
167 switch ( sumOfDice )
168 {
169     // win on first roll
170     case LUCKY_SEVEN:
171     case YO_LEVEN:
172         resultJTextField.setText( "You win!!!" );
173         break;
174
175     // lose on first roll
176     case SNAKE_EYES:
177     case TREY:
178     case BOX_CARS:
179         resultJTextField.setText( "Sorry, you lose." );
180         break;
181
182     // remember point in instance variable
183     default:
184
185         // set the point and output result
186         myPoint = sumOfDice;
187         resultJTextField.setText( "Roll again!" );
188
189         // show the dice images
190         pointDie1JLabel.setIcon( die1JLabel.getIcon() );
191         pointDie2JLabel.setIcon( die2JLabel.getIcon() );
192
193         // update the border title
194         pointDiceTitledBorder.setTitle(
195             "Point is " + sumOfDice );
196         pointDiceJPanel.repaint();
197
198         // change the state of the JButtons
199         playJButton.setEnabled( false );
200         rollJButton.setEnabled( true );
201
202     } // end switch statement
203
204 } // end method playJButtonActionPerformed
205
206 // continue the game
207 private void rollJButtonActionPerformed( ActionEvent event )
208 {
209     int sumOfDice = rollDice(); // roll dice
210
211     // determine outcome of roll, player matches point
212     if ( sumOfDice == myPoint )
213     {
214         resultJTextField.setText( "You win!!!" );
215         rollJButton.setEnabled( false );
216         playJButton.setEnabled( true );
217     }
218     // determine outcome of roll, player loses
219     else if ( sumOfDice == CRAPS )
220     {
221         resultJTextField.setText( "Sorry, you lose" );
222         rollJButton.setEnabled( false );

```

Winning on the first roll

Losing on the first roll

Player must match the point

Displaying die images

Displaying point and updating the JPanel

Allowing the player to roll again

Figure 15.25 Craps Game application code listing. (Part 4 of 5.)

```

223     playJButton.setEnabled( true );
224 }
225
226 } // end method rollJButtonActionPerformed
227
228 // generate random die rolls
229 private int rollDice()
230 {
231     // generate random die values
232     int die1 = 1 + randomObject.nextInt( 6 );
233     int die2 = 1 + randomObject.nextInt( 6 );
234
235     // display the dice images
236     displayDie( die1JLabel, die1 );
237     displayDie( die2JLabel, die2 );
238
239     return die1 + die2; // return sum of dice values
240
241 } // end method rollDice
242
243 // displays the die image
244 private void displayDie( JLabel picDieJLabel, int face )
245 {
246     ImageIcon image =
247         new ImageIcon( FILE_PREFIX + face + FILE_SUFFIX );
248
249     // display die images in picDieJLabel
250     picDieJLabel.setIcon( image );
251
252 } // end method displayDie
253
254 // main method
255 public static void main( String args[] )
256 {
257     CrapsGame application = new CrapsGame();
258     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
259
260 } // end method main
261
262 } // end class CrapsGame

```

Generating random numbers —

Displaying the image in the JLabel —

Figure 15.25 Craps Game application code listing. (Part 5 of 5.)

SELF-REVIEW

- You request components in a JPanel to be redisplayed by calling the _____ method.
 - paint
 - update
 - repaint
 - redraw
- To clear the image in a JLabel, set its *icon* property to _____.
 - "" (double quotes)
 - null
 - none
 - 0

Answers: 1) c. 2) b.

15.5 Wrap-Up

In this tutorial, you created the **Craps Game** application to simulate playing the popular dice game Craps. You learned about the `Random` class and how it can be used to generate random numbers by creating a `Random` object and calling its `next-`

Int method. You then learned how to specify the range of values within which random numbers should be generated by passing an argument to the `nextInt` method. You also learned that the `Random` class can be used to generate random doubles using the `nextDouble` method. Later, you learned about borders and how to set the border of a component.

Using your knowledge of random number generation and methods, you wrote code that added functionality to your **Craps Game** application. You used random-number generation to simulate the element of chance, and you learned how to use code to display an image in a `JLabel`.

In the next tutorial, you will learn how to use arrays, which allow you to use one name to store many values. You will apply your knowledge of random numbers and arrays to create a **Flag Quiz** application that tests a user's knowledge of various national flags.

SKILLS SUMMARY

Generating Random Numbers

- Create an object of class `Random` and call the object's `nextInt` method.

Generating Random Numbers within a Specified Range

- Call the `Random` class's `nextInt` method. Use an argument to specify a range of random numbers from zero to one less than the argument. Add a number to this value to shift the range.
- Call the `Random` class's `nextDouble` method. Multiply a number by the return value to change the size of the range. Add a number to this value to shift the range.

Adding a `TitledBorder` to a Component

- Use the `setBorder` method and pass to it a `TitledBorder` object to add a title and border to a component.
-

KEY TERMS

- absolute positioning**—Specifies the exact size and location of a component in a container.
- border property**—Allows a border to be added to some GUI components.
- import declaration**—Used to import classes or packages.
- java.util package**—Provides, among other capabilities, random number generation capabilities with class `Random`.
- JPanel component**—Groups related components.
- layout property**—Controls how components are arranged in a container (such as the content pane or `JPanel`).
- method overloading**—Allows several methods to have the same name but different numbers of arguments or different types of arguments.
- new keyword**—Creates an object and assigns it a location in memory.
- nextDouble method of Random**—Generates a random positive double value that is greater than or equal to 0.0 and less than 1.0.
- nextInt method of Random**—Generates an `int` value selected from all possible `int` values, when called with no arguments. When called with an `int` argument, it generates an `int` value from 0 to one less than the argument.
- null keyword**—Value that clears a reference.
- package**—A group of related classes. A package can be imported to add functionality to an application.
- Random class**—Contains methods to generate random numbers. Declared in package `java.util`.
- reference**—A variable that refers to an object. A reference specifies the location in the computer's memory of an object.
- relative positioning**—Specifies the size and location of components in relation to other components on a container.
- repaint method of JPanel**—Redisplays the `JPanel` as soon as possible.

setBorder method of JPanel—Sets the border that is displayed around the JPanel.
setLayout method of JPanel—Sets the way components are arranged in the JPanel.
setTitle method of TitledBorder—Sets the text displayed in the TitledBorder.
title property of TitledBorder—Controls the text that is displayed in the border.
TitledBorder class—Allows a component to have a border with a String title.

JAVA LIBRARY REFERENCE

Random This class is used to generate random numbers.

■ Methods

nextInt—Generates an `int` value selected from all possible `int` values, when called with no arguments. When called with an `int` argument, it generates an `int` value from 0 to one less than that argument.
nextDouble—Generates a positive `double` value that is greater than or equal to 0.0 and less than 1.0.

JPanel This component groups other related components. This grouping allows a single border to be placed around multiple components. Components placed in a JPanel are positioned with respect to the upper-left corner of the JPanel, so changing the location of the JPanel moves all of the components contained inside it.

■ Methods

add—Adds a component to the JPanel.
setBorder—Sets the border displayed around the JPanel.
setBounds—Sets the *bounds* property, which specifies the location and size of a JPanel.
setLayout—Sets the *layout* property, which controls how components are displayed on the JPanel.

TitledBorder This class allows the user to add a String title and line border to a component.

■ Methods

setTitle—Sets the title that is displayed in the border.

MULTIPLE-CHOICE QUESTIONS

15.1 A Random object can generate random numbers of type _____.

- a) `int`
- b) `String`
- c) `double`
- d) Both a and c.

15.2 Import declarations allow the application access to _____ from the Java class library.

- a) packages and classes
- b) classes and objects
- c) objects and methods
- d) methods and variables

15.3 The _____ method sets the title of a TitledBorder.

- a) `title`
- b) `setText`
- c) `setBorder`
- d) `setTitle`

15.4 The `nextInt` method of class Random can be called using _____.

- a) one argument
- b) no arguments
- c) two arguments
- d) Both a and b.

15.5 The statement _____ assigns to `value` a random number in the range 5 to 20.

- a) `value = 4 + randomObject.nextInt(16);`
- b) `value = randomObject.nextInt(21);`
- c) `value = 5 + randomObject.nextInt(15);`
- d) `value = 5 + randomObject.nextInt(16);`

- 15.6** The _____ method displays an image on a JLabel.
- setImage
 - setIcon
 - setImageIcon
 - None of the above.
- 15.7** The java.util package contains class Random to _____.
- generate positive integers
 - generate positive doubles
 - provide random number generation capabilities
 - All of the above.
- 15.8** Assume that randomGenerator is an object of class Random. The expression randomGenerator.nextDouble() produces random numbers in the range _____.
- 0.0 to less than 1.0
 - greater than 0.0 to less than 1.0
 - 0.0 to 1.0
 - greater than 0.0 to 1.0
- 15.9** When creating random numbers, the argument passed to the nextInt method needs to be _____.
- equal to the maximum value you wish to generate
 - equal to one more than the maximum value you wish to generate
 - equal to one less than the maximum value you wish to generate
 - equal to the minimum value you wish to generate
- 15.10** When a variable is used throughout the lifetime of the application and its value will not change, it should be declared as a(n)_____.
- local variable
 - constant
 - instance variable
 - None of the above.

EXERCISES

- 15.11 (Guess the Number Application)** Develop an application that generates a random number and prompts the user to guess the number as in Fig. 15.26. When the user clicks the **New Game** JButton, the application chooses a number in the range from 1 to 100 at random. The user enters guesses into the **Guess:** JTextField and clicks the **Enter** JButton. If the guess is correct, the game ends, and the user can start a new game. If the guess is not correct, the application should indicate if the guess is higher or lower than the correct number.

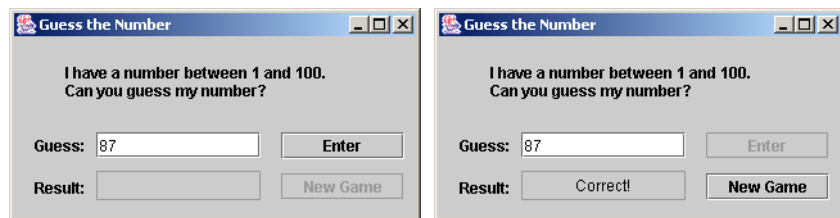


Figure 15.26 Guess the Number application.

- Copying the template to your working directory.** Copy the directory C:\Examples\Tutorial15\Exercises\GuessNumber to your C:\SimplyJava directory.
- Opening the template file.** Open the GuessNumber.java file in your text editor.
- Creating a Random object.** In line 28, create two instance variables. The first variable should store a Random object and the second variable should store a randomly generated number in the range 1 to 100 created using the Random object.
- Adding code to the enterJButtonActionPerformed method.** Add code starting in line 133 to the enterJButtonActionPerformed method that retrieves the value entered by the user in guessJTextField and compares that value to the randomly generated number. If the user's guess is lower than the correct answer, display **Too low...** in resultJTextField. If the user's guess is higher than the correct answer, display **Too high...** in resultJTextField. If the guess is correct, display **Correct!** in resultJTextField. Then disable the **Enter** JButton and enable the **New Game** JButton.

- e) **Adding code to the newGameJButtonActionPerformed method.** Add code to the newGameJButtonActionPerformed method (after the enterJButtonActionPerformed method) that clears resultJTextField and generates a new random number for the instance variable. The method should then disable the **New Game** JButton and enable the **Enter** JButton.
- f) **Saving the application.** Save your modified source code file.
- g) **Opening the Command Prompt window and changing directories.** Open the **Command Prompt** by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\GuessNumber`.
- h) **Compiling the application.** Compile your application by typing `javac GuessNumber.java`
- i) **Running the application.** When your application compiles correctly, run it by typing `java GuessNumber`. Test your application by playing the game until you guess the correct number. Then click **New Game** and play once more.
- j) **Closing the application.** Close your running application by clicking its close button.
- k) **Closing the Command Prompt window.** Close the **Command Prompt** window by clicking its close button.

15.12 (Dice Simulator Application) Develop an application that simulates rolling two six-sided dice. Your application should have a **Roll** JButton that, when clicked, displays two die faces (images) corresponding to random numbers. It should also display the number of times each face has appeared. Your application should appear similar to Fig. 15.27. This application will help you see if rolling dice on your computer is really random. If it is, the number of 1s, 2s, 3s, 4s, 5s and 6s you roll should be about the same, at least for a large number of rolls. The **Total**: JTextField should hold the number of rolls of the dice (each time the **Roll** JButton is clicked, the value in the **Total**: JTextField should be incremented by 2).

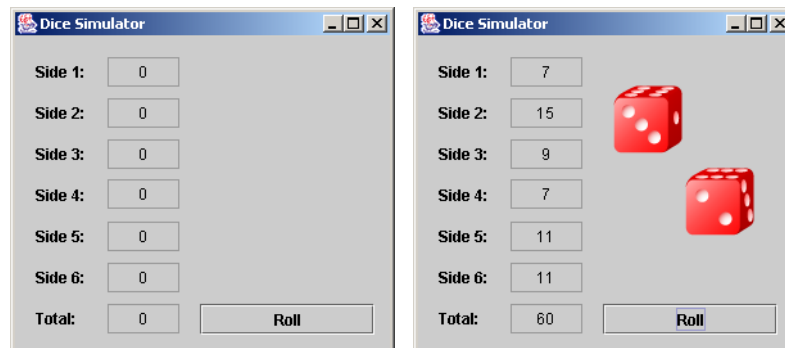


Figure 15.27 Dice Simulator application.

- a) **Copying the template to your working directory.** Copy the directory `C:\Examples\Tutorial15\Exercises\DiceSimulator` to your `C:\SimplyJava` directory.
- b) **Opening the template file.** Open the `DiceSimulator.java` file in your text editor.
- c) **Displaying the die image.** After the `rollJButtonActionPerformed` method, in line 211, create a method named `displayDie` that takes a `JLabel` component as its argument. This method should create a new random integer from 1 to 6 using the `nextInt` method of the `Random` object generator and assign that value to the variable `face`. Display the die image in the `JLabel` component that was passed as an argument. The die image should correspond to the random number that was generated. To set the image, refer to the code presented in Fig. 15.25. Finally, this method should call the `displayFrequency` method to display the number of times each face has occurred.
- d) **Coding the rollJButtonActionPerformed method.** Add code to the `rollJButtonActionPerformed` method to call the `displayDie` method twice to display the images for `die1JLabel` and `die2JLabel`. Add code to increment the value displayed in the **Total**: JTextField by 2.
- e) **Displaying the frequency.** After the `displayDie` method, Create a method named `displayFrequency` that takes an `int` representing a die roll as its argument. This

method will use a `switch` statement with the `int` parameter as the control variable. Each case should update the number of times its corresponding face has appeared.

- f) **Saving the application.** Save your modified source code file.
- g) **Opening the Command Prompt window and changing directories.** Open the **Command Prompt** by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\DiceSimulator`.
- h) **Compiling the application.** Compile your application by typing `javac DiceSimulator.java`.
- i) **Running the application.** When your application compiles correctly, run it by typing `java DiceSimulator`. Test your application by clicking the **Roll** JButton 30 times (to roll 60 dice). Make sure that the total number of rolls adds up to 60.
- j) **Closing the application.** Close your running application by clicking its close button.
- k) **Closing the Command Prompt window.** Close the **Command Prompt** window by clicking its close button.

15.13 (Lottery Picker Application) A lottery commission offers four different lottery games to play: Three number, Four number, Five number and Five number + 1 lotteries. Each game has independent numbers. Develop an application that randomly picks numbers for all four games and displays the generated numbers in a GUI (Fig. 15.28). The games are played as follows:

- Three number lotteries require players to choose three numbers in the range from 0–9.
- Four number lotteries require players to choose four numbers, in the range from 0–9.
- Five number lotteries require players to choose five numbers in the range from 1–39.
- Five number + 1 lotteries require players to choose five numbers in the range 1–49 and an additional number in the range from 1–42.

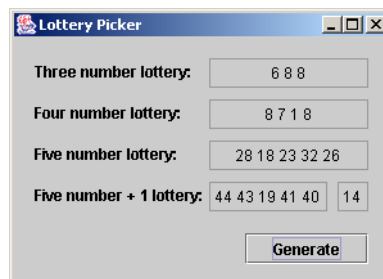


Figure 15.28 Lottery Picker application.

- a) **Copying the template to your working directory.** Copy the directory `C:\Examples\Tutorial15\Exercises\LotteryPicker` to your `C:\SimplyJava` directory.
- b) **Opening the template file.** Open the `LotteryPicker.java` file in your text editor.
- c) **Generating random numbers.** Create a method in line 142 named `generate` that will take two `ints`, representing the low and high end of a range of random numbers, and return a `String` containing a generated random number.
- d) **Drawing numbers for the games.** Add code to the `generateJButtonActionPerformed` method to call the `generate` method and display the generated numbers for all four games. Some lotteries allow repetition of numbers. To make this application simple, allow repetition of numbers for all the lotteries.
- e) **Saving the application.** Save your modified source code file.
- f) **Opening the Command Prompt window and changing directories.** Open the **Command Prompt** by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\Lottery-Picker`.
- g) **Compiling the application.** Compile your application by typing `javac LotteryPicker.java`.

- h) **Running the application.** When your application compiles correctly, run it by typing `java LotteryPicker`. Test your application by clicking the **Generate** JButton. Make sure that the resulting lottery numbers are within the bounds given in the rules above.
- i) **Closing the application.** Close your running application by clicking its close button.
- j) **Closing the Command Prompt window.** Close the **Command Prompt** window by clicking its close button.

What does this code do? ▶ **15.14** This code displays text in `integer1JTextField`, `double1JTextField`, and `integer2JTextField`. What is displayed in these JTextFields?

```

1 private void pickRandomNumbers()
2 {
3     Random randomObject = new Random();
4
5     int number1 = randomObject.nextInt();
6     double number = 5 * randomObject.nextDouble();
7     int number2 = 1 + randomObject.nextInt( 11 );
8     integer1JTextField.setText = String.valueOf( number1 );
9     double1JTextField.setText = String.valueOf( number );
10    integer2JTextField.setText = String.valueOf( number2 );
11
12 } // end method pickRandomNumbers

```

What's wrong with this code? ▶ **15.15** This `randomDecimal` method should assign a random `double` number (in the range 0.0 to less than 50.0) to `double number` and display it in `displayJLabel`. Find the error(s) in the following code.

```

1 private void randomDecimal()
2 {
3     double number;
4     Random randomObject = new Random();
5
6     number = randomObject.nextDouble();
7     displayJLabel.setText = String.valueOf( number );
8
9 } // end method randomDecimal

```

Programming Challenge ▶ **15.16 (Multiplication Teacher Application)** Develop an application that helps children learn multiplication as in Fig. 15.29. Use random-number generation to produce two positive one-digit integers that display in a question, such as “How much is 6 times 7?” The student should type the answer into a `JTextField`. If the answer is correct, then the application randomly displays one of three messages: **Very Good!**, **Excellent!** or **Great Job!** in a `JLabel` and displays the next question. If the student is wrong, the `JLabel` displays the message **No. Please try again.**

- a) **Copying the template to your working directory.** Copy the directory `C:\Examples\Tutorial15\Exercises\MultiplicationTeacher` to your `C:\SimplyJava` directory.
- b) **Opening the template file.** Open the `MultiplicationTeacher.java` file in your text editor.
- c) **Generating the questions.** Declare a method named `generateQuestion` in your application to generate and display each new question in `questionJLabel`.
- d) **Adding a call to the generateQuestion method.** Add a call to the `generateQuestion` method at the end of the `createUserInterface` method.

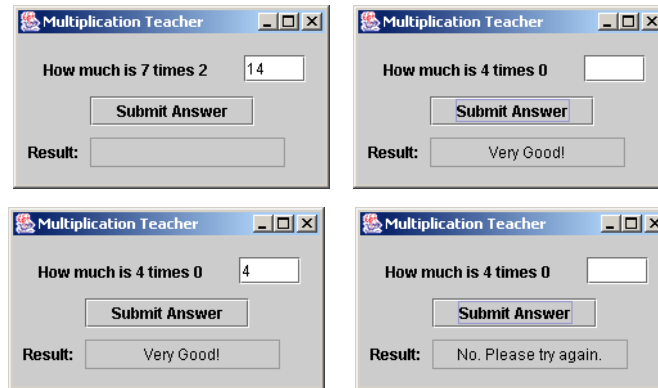


Figure 15.29 Multiplication Teacher application.

- e) **Displaying a random message.** Add a method named `generateOutput` that displays a random message congratulating the student if they answer correctly.
- f) **Determining whether the right answer was entered.** Add code to the `submit-JButtonActionPerformed` method declared in your application. Determine whether the student answered the question correctly and display an appropriate message. If the student answered the question correctly, call the `generateOutput` method, then call the `generateQuestion` method. Otherwise, display a message indicating the user is wrong. After displaying either result, clear `answerJTextField`.
- g) **Saving the application.** Save your modified source code file.
- h) **Opening the Command Prompt window and changing directories.** Open the **Command Prompt** window by selecting **Start > Programs > Accessories > Command Prompt**. Change to your working directory by typing `cd C:\SimplyJava\MultiplicationTeacher`.
- i) **Compiling the application.** Compile your application by typing `javac MultiplicationTeacher.java`.
- j) **Running the application.** When your application compiles correctly, run it by typing `java MultiplicationTeacher`. Test your application by answering the questions and clicking the **Submit Answer** **JButton**. Make sure that if you continue answering questions, you will see all four possible result messages.
- k) **Closing the application.** Close your running application by clicking its close button.
- l) **Closing the Command Prompt window.** Close the **Command Prompt** window by clicking its close button.