Chapter 3: Using Classes and Objects
Lab Exercises

Topics	Lab Exercises
String Class	Prelab Exercises Working with Strings
Random Class	Rolling Dice
Math Class	Computing Distance
Formatting Classes	Formatting Output
Enumerated Types	Playing with Cards
Wrapper Classes	Experimenting with the Integer Class
Panels	Nested Panels

Prelab Exercises
Sections 3.2-3.5

These exercises focus on the String, Random, and Math classes defined in the Java Standard Class Library. The main concepts are in the text in sections 3.2-3.5. The goals of the lab are for you to gain experience with the following concepts:

- Declaring a variable to be a reference to an object—for example, the following declares the variable *quotation* to be a reference to a String object:

```
String quotation;
```

- Declaring a variable to be a reference to an object and creating the object (*instantiating* it) using the *new* operator—for example,

```
String quotation = new String("I think, therefore I am.");
Random generator = new Random();
```

- Invoking a method to operate on an object using the *dot operator*—for example,

```
quotation.length()
```

  invokes the length method which returns the length of the quotation String or

```
quotation.toLowerCase()
```

  quotation except all letters are lower case. These invocations would be used in a program in a place appropriate for an integer (in the first case) or a String (in the second case) such as an assignment statement or a println statement.

- Invoking *static* or *class* methods—these are methods that are invoked using the class name rather than an object name. The methods in the Math class are static methods (basically because we don't need different instances of Math whereas there are lots of different String objects). Examples are

```
Math.sqrt(2)    (which returns the square root of 2)
```

  and

```
Math.pow(3, 2)  (which returns 3^2)
```

- Importing the appropriate packages—usually when you use classes from a library you need to put the *import* declaration at the top of your program. The exception is for classes defined in the java.lang package (this includes String and Math) which is automatically imported into every Java program.

Exercises

1. Fill in the blanks in the program below as follows: (Section 3.2, especially the example in Listing 3.1, should be helpful):
   (a) declare the variable *town* as a reference to a String object and initialize it to "Anytown, USA".
   (b) write an assignment statement that invokes the *length* method of the string class to find the length of the *college* String object and assigns the result to the stringLength variable
   (c) complete the assignment statement so that *change1* contains the same characters as *college* but all in upper case
   (d) complete the assignment statement so that *change2* is the same as *change1* except all capital O's are replaced with the asterisk (*) character.
   (e) complete the assignment statement so that *change3* is the concatenation of *college* and *town* (use the *concat* method of the String class rather than the + operator)

```
// ************************************************
//   StringPlay.java
//
//   Play with String objects
// ************************************************
public class StringPlay
{
   public static void main (String[] args)
   {
      String college = new String ("PoDunk College");

      _____________________________________________________________; // part (a)

      int stringLength;
      String change1, change2, change3;

      _____________________________________________________________; // part (b)

      System.out.println (college + " contains " + stringLength + " characters.");

      change1 = ___________________________________________________; // part (c)

      change2 = ___________________________________________________; // part (d)

      change3 = ___________________________________________________; // part (e)

      System.out.println ("The final string is " + change3);
    }
}
```

2. The following program should read in the lengths of two sides of a right triangle and compute the length of the hypotenuse (recall that the length of the hypotenuse is the square root of side 1 squared plus side 2 squared). Complete it by adding statements to read the input from the keyboard and to compute the length of the hypotenuse (you need to use a Math class method for that).

```
// ********************************************************************
//   RightTriangle.java
//
//   Compute the length of the hypotenuse of a right triangle
//   given the lengths of the sides
// ********************************************************************
import java.util.Scanner;

public class RightTriangle
{
   public static void main (String[] args)
   {
      double side1, side2;  // lengths of the sides of a right triangle
      double hypotenuse;    // length of the hypotenuse

      Scanner scan = new Scanner(System.in);

      // Get the lengths of the sides as input
      System.out.print ("Please enter the lengths of the two sides of " +
                        "a right triangle (separate by a blank space): ");

      _____________________________________________________________;

      _____________________________________________________________;

      // Compute the length of the hypotenuse

      _____________________________________________________________;

      // Print the result
      System.out.println ("Length of the hypotenuse: " + hypotenuse);
    }
}
```

3. In many situations a program needs to generate a random number in a certain range. The Java Random class lets the programmer create objects of type Random and use them to generate a stream of random numbers (one at a time). The following declares the variable *generator* to be an object of type Random and instantiates it with the *new* operator.

```
Random generator = new Random();
```

The *generator* object can be used to generate either integer or floating point random numbers using either the *nextInt* method (either with no parameter or with a single integer parameter) or *nextFloat* (or *nextDouble)* methods, respectively. The integer returned by *nextIn(t* could be any valid integer (positive or negative) whereas the number returned by *nextInt(n)* is a random integer in the range 0 to n-1. The numbers returned by *nextFloat()* or *nextDouble()* are floating point numbers between 0 and 1 (up to but not including the 1). Most often the goal of a program is to generate a random integer in some particular range, say 30 to 99 (inclusive). There are several ways to do this:

- **Using nextInt():** This way we must use the % operator to reduce the range of values—for example,

```
Math.abs(generator.nextInt()) % 70
```

will return numbers between 0 and 69 (because those are the only possible remainders when an integer is divided by 70 - note that the absolute value of the integer is first taken using the *abs* method from the Math class). In general, using % N will give numbers in the range 0 to N - 1. Next the numbers must be shifted to the desired range by adding the appropriate number. So, the expression

```
Math.abs(generator.nextInt()) % 70 + 30
```

will generate numbers between 30 and 99.

- **Using nextInt(70):** The expression

  ```
  generator.nextInt(70)
  ```

  will return numbers between 0 and 69 (inclusive). Next the numbers must be shifted to the desired range by adding the appropriate number. So, the expression

  ```
  generator.nextInt(70) + 30
  ```

  will generate numbers between 30 and 99.

- **Using nextFloat:** In this case, we must multiply the result of nextFloat to expand the range—for example,

  ```
  generator.nextFloat() * 70
  ```

  returns a floating point number between 0 and 70 (up to but not including 70). To get the integer part of the number we use the cast operator:

  ```
  (int) (generator.nextFloat() * 70)
  ```

  The result of this is an integer between 0 and 69, so

  ```
  (int) (generator.nextFloat() * 70) + 30
  ```

  shifts the numbers by 30 resulting in numbers between 30 and 99.

  The method *nextFloat* can be replaced by *nextDouble* to get double precision floating point numbers rather than single precision.

Fill in the blanks in the following program to generate the random numbers as described in the documentation. NOTE that that java.util.Random must be imported to use the Random class.

```
// ************************************************
//   LuckyNumbers.java
//
//   To generate three random "lucky" numbers
// ************************************************

import java.util.Random;

public class LuckyNumbers
{
   public static void main (String[] args)
   {
      Random generator = new Random();
      int lucky1, lucky2, lucky3;

      // Generate lucky1 (a random integer between 50 and 79) using the
      // nextInt method (with no parameter)

      lucky1 = ____________________________________________________________;

      // Generate lucky2 (a random integer between 90 and 100) using the
      // nextInt method with an integer parameter

      lucky2 = ____________________________________________________________;
```

```
        // Generate lucky3 (a random integer between 11 and 30) using nextFloat

        lucky3 = ____________________________________________________________;

        System.out.println ("Your lucky numbers are " + lucky1 + ", " + lucky2
                            + ", and " + lucky3);
    }
}
```

Working with Strings

The following program illustrates the use of some of the methods in the String class. Study the program to see what it is doing.

```
// ********************************************************************
//   StringManips.java
//
//   Test several methods for manipulating String objects
// ********************************************************************

import java.util.Scanner;

public class StringManips
{
    public static void main (String[] args)
    {
      String phrase = new String ("This is a String test.");
      int phraseLength;   // number of characters in the phrase String
      int middleIndex;    // index of the middle character in the String
      String firstHalf;   // first half of the phrase String
      String secondHalf;  // second half of the phrase String
      String switchedPhrase; // a new phrase with original halves switched

      // compute the length and middle index of the phrase
      phraseLength = phrase.length();
      middleIndex = phraseLength / 2;

      // get the substring for each half of the phrase
      firstHalf = phrase.substring(0,middleIndex);
      secondHalf = phrase.substring(middleIndex, phraseLength);

      // concatenate the firstHalf at the end of the secondHalf
      switchedPhrase = secondHalf.concat(firstHalf);

      // print information about the phrase
      System.out.println();
      System.out.println ("Original phrase: " + phrase);
      System.out.println ("Length of the phrase: " + phraseLength +
                          " characters");
      System.out.println ("Index of the middle: " + middleIndex);
      System.out.println ("Character at the middle index: " +
                          phrase.charAt(middleIndex));
      System.out.println ("Switched phrase: " + switchedPhrase);

      System.out.println();
    }
}
```

The file *StringManips.java* contains this program. Save the file to your directory and compile and run it. Study the output and make sure you understand the relationship between the code and what is printed. Now modify the file as follows:

1. Declare a variable of type String named *middle3* (put your declaration with the other declarations near the top of the program) and use an assignment statement and the *substring* method to assign *middle3* the substring consisting of the middle three characters of *phrase* (the character at the middle index together with the character to the left of that and the one to the right – use variables, not the literal indices for this particular string). Add a println statement to print out the result. Save, compile, and run to test what you have done so far.

2. Add an assignment statement to replace all blank characters in *switchedPhrase* with an asterisk (*). The result should be stored back in *switchedPhrase* (so switchedPhrase is actually changed). (Do not add another print—place your statement in the program so that this new value of *switchedPhrase* will be the one printed in the current println statement.) Save, compile, and run your program.

3. Declare two new variables *city* and *state* of type String. Add statements to the program to prompt the user to enter their hometown—the city and the state. Read in the results using the appropriate Scanner class method – you will need to have the user enter city and state on separate lines. Then using String class methods create and print a new string that consists of the state name (all in uppercase letters) followed by the city name (all in lowercase letters) followed again by the state name (uppercase). So, if the user enters Lilesville for the city and North Carolina for the state, the program should create and print the string

```
NORTH CAROLINAlilesvilleNORTH CAROLINA
```

Rolling Dice

Write a complete Java program that simulates the rolling of a pair of dice. For each die in the pair, the program should generate a random number between 1 and 6 (inclusive). It should print out the result of the roll for each die and the total roll (the sum of the two dice), all appropriately labeled. You must use the Random class. The RandomNumbers program in listing 3.2 of the text may be helpful.

Computing Distance

The file *Distance.java* contains an incomplete program to compute the distance between two points. Recall that the distance between the two points (x1, y1) and (x2, y2) is computed by taking the square root of the quantity $(x1 - x2)^2 + (y1 - y2)^2$. The program already has code to get the two points as input. You need to add an assignment statement to compute the distance and then a print statement to print it out (appropriately labeled). Test your program using the following data: The distance between the points (3, 17) and (8, 10) is 8.6023... (lots more digits printed); the distance between (–33, 49) and (–9, –15) is 68.352.…

```
// ********************************************************
//   Distance.java
//
//   Computes the distance between two points
// ********************************************************

import java.util.Scanner;

public class Distance
{
    public static void main (String[] args)
    {
      double x1, y1, x2, y2; // coordinates of two points
      double distance;       // distance between the points

      Scanner scan = new Scanner(System.in);

      // Read in the two points
      System.out.print ("Enter the coordinates of the first point " +
                "(put a space between them): ");
      x1 = scan.nextDouble();
      y1 = scan.nextDouble();

      System.out.print ("Enter the coordinates of the second point: ");
      x2 = scan.nextDouble();
      y2 = scan.nextDouble();

      // Compute the distance

      // Print out the answer

    }
}
```

Formatting Output

File *Deli.java* contains a partial program that computes the cost of buying an item at the deli. Save the program to your directory and do the following:

1. Study the program to understand what it does.

2. Add the import statements to import the DecimalFormat and NumberFormat classes.

3. Add the statement to declare *money* to be a NumberFormat object as specified in the comment.

4. Add the statement to declare *fmt* to be a DecimalFormat object as specified in the comment.

5. Add the statements to print a label in the following format (the numbers in the example output are correct for input of $4.25 per pound and 41 ounces). Use the formatting object money to print the unit price and total price and the formatting object fmt to print the weight to 2 decimal places.

   ***** CS Deli *****

   Unit Price: $4.25 per pound
   Weight: 2.56 pounds

   TOTAL: $10.89

```
// ********************************************************
// DeliFormat.java
//
// Computes the price of a deli item given the weight
// (in ounces) and price per pound -- prints a label,
// nicely formatted, for the item.
//
// ********************************************************

import java.util.Scanner;



public class Deli
{
    // -------------------------------------------------
    //  main reads in the price per pound of a deli item
    //  and number of ounces of a deli item then computes
    //  the total price and prints a "label" for the item
    //  -------------------------------------------------

    public static void main (String[] args)
    {
        final double OUNCES_PER_POUND = 16.0;

        double pricePerPound;  // price per pound
        double weightOunces;   // weight in ounces
        double weight;               // weight in pounds
        double totalPrice;       // total price for the item

        Scanner scan = new Scanner(System.in);

        // Declare money as a NumberFormat object and use the
        // getCurrencyInstance method to assign it a value


        // Declare fmt as a DecimalFormat object and instantiate
        // it to format numbers with at least one digit to the left of the
        // decimal and the fractional part rounded to two digits.


        // prompt the user and read in each input
        System.out.println ("Welcome to the CS Deli!!\n ");

        System.out.print ("Enter the price per pound of your item: ");
        pricePerPound = scan.nextDouble();

        System.out.print ("Enter the weight (ounces): ");
        weightOunces = scan.nextDouble();

        // Convert ounces to pounds and compute the total price
        weight = weightOunces / OUNCES_PER_POUND;
        totalPrice = pricePerPound * weight;

        // Print the label using the formatting objects
        // fmt for the weight in pounds and money for the prices



    }
}
```

Playing With Cards

Write a class that defines an enumerated type named *Rank* with values `ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen, king.` The main method should do the following:

1. Declare variables *highCard*, *faceCard*, *card1*, and *card2* of type Rank.

2. Assign highCard to be an ace, faceCard to be a jack, queen or king (your choice), and card1 and card2 to be two different numbered cards (two through ten - your choice).

3. Print a line, using the highCard and faceCard objects, in the following format:

   ```
   A blackjack hand: ace and .....
   ```

   The faceCard variable should be printed instead of the dots.

4. Declare two variables card1Val and card2Val of type int and assign them the face value of your card1 and card2 objects. Use your card1 and card2 variables and the ordinal method associated with enumerated types. Remember that the face value of two is 2, three is 3, and so on so you need to make a slight adjustment to the ordinal value of the enumerated type.

5. Print two lines, using the card1 and card2 objects and the name method, as follows:

   ```
   A two card hand:   (print card1 and card2)
   Hand value:  (print the sum of the face values of the two cards)
   ```

Experimenting with the Integer Class

Wrapper classes are described on pages 138-140 of the text. They are Java classes that allow a value of a primitive type to be "wrapped up" into an object, which is sometimes a useful thing to do. They also often provide useful methods for manipulating the associated type. Wrapper classes exist for each of the primitive types: boolean, char, float, double, int, long, short, and byte.

Write a program IntWrapper that uses the constants and methods of the Integer class (page 140 for a short list, pages 819-820 for a complete list) to perform the following tasks. Be sure to clearly label your output and test your code for each task before proceding.

1. Prompt for and read in an integer, then print the binary, octal and hexadecimal representations of that integer.

2. Print the maximum and minimum possible Java integer values. Use the constants in the Integer class that hold these values -- don't type in the numbers themselves. Note that these constants are static (see the description on page 140 and the signature on page 819).

3. Prompt the user to enter two decimal integers, one per line. Use the *next* method of the Scanner class to read each of them in. (The *next* method returns a String so you need to store the values read in String variables, which may seem strange.) Now convert the strings to ints (use the appropriate method of the Integer class to do this), add them together, and print the sum.

Nested Panels

The program NestedPanels.java is from Listing 3.8 of the text. Save the program to your directory and do the following:

1. Compile and run the program. Experiment with resizing the frame and observe the effect on the components.
2. Modify the program by adding a third subpanel that is twice as wide, but the same height, as the other two subpanels. Choose your own label and color for the subpanel (the color should not be red, green, or blue). Add the panel to the primary panel after the other two panels.
3. Compile and run the modified program. Again, experiment with resizing the frame and observe the effect on the components.
4. Now add a statement to the program to set the preferred size of the primary panel to 320 by 260. (What would be the purpose of this?). Compile and run the program to see if anything changed.
5. Now add another panel with background color blue and size 320 by 20. Add a "My Panels" label to this panel and then add this panel to the primary panel before adding the other panels. Compile and run the program. What was the effect of this panel?

```
//********************************************************************
//  NestedPanels.java       Author: Lewis/Loftus
//
//  Demonstrates a basic component hierarchy.
//********************************************************************
import java.awt.*;
import javax.swing.*;

public class NestedPanels
{
   //-----------------------------------------------------------------
   //  Presents two colored panels nested within a third.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Nested Panels");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      // Set up first subpanel
      JPanel subPanel1 = new JPanel();
      subPanel1.setPreferredSize (new Dimension(150, 100));
      subPanel1.setBackground (Color.green);
      JLabel label1 = new JLabel ("One");
      subPanel1.add (label1);

      // Set up second subpanel
      JPanel subPanel2 = new JPanel();
      subPanel2.setPreferredSize (new Dimension(150, 100));
      subPanel2.setBackground (Color.red);
      JLabel label2 = new JLabel ("Two");
      subPanel2.add (label2);

      // Set up primary panel
      JPanel primary = new JPanel();
      primary.setBackground (Color.blue);
      primary.add (subPanel1);
      primary.add (subPanel2);

      frame.getContentPane().add(primary);
      frame.pack();
      frame.setVisible(true);
   }
}
```