PROGRAMMING EXERCISES

★ ★ **Exercise P8.1.** Implement the `Coin` class described in Section 8.2. Modify the `CashRegister` class so that coins can be added to the cash register, by supplying a method

```
void enterPayment(int coinCount, Coin coinType)
```

The caller needs to invoke this method multiple times, once for each type of coin that is present in the payment.

Use the following class as your tester class:

```
/**
   This program tests the CashRegister class.
*/
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      final double NICKEL_VALUE = 0.05;
      final double DIME_VALUE = 0.1;
      final double QUARTER_VALUE = 0.25;

      CashRegister myRegister = new CashRegister();
      myRegister.recordPurchase(0.82);
      myRegister.enterPayment(3, new Coin(QUARTER_VALUE, "quarter"));
      myRegister.enterPayment(2, new Coin(NICKEL_VALUE, "nickel"));

      double myChange = myRegister.giveChange();
      System.out.println("Change: " + myChange);
      System.out.println("Expected: 0.03");
   }
}
```

★ ★ **Exercise P8.2.** Modify the `giveChange` method of the `CashRegister` class so that it returns the number of coins of a particular type to return:

```
int giveChange(Coin coinType)
```

The caller needs to invoke this method for each coin type, in decreasing value. Use the following class as your tester class:

```
/**
   This program tests the CashRegister class.
*/
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      final double PENNY_VALUE = 0.01;
      final double NICKEL_VALUE = 0.05;
      final double DIME_VALUE = 0.1;
      final double QUARTER_VALUE = 0.25;

      CashRegister myRegister = new CashRegister();
      myRegister.recordPurchase(0.82);
      myRegister.enterPayment(1.00);

      Coin[] CoinTypes = new Coin[]
      {
         new Coin(QUARTER_VALUE, "quarter"),
```

```
            new Coin(DIME_VALUE, "dime"),
            new Coin(NICKEL_VALUE, "nickel"),
            new Coin(PENNY_VALUE, "penny")
         };
         int[] expected = { 0, 1, 1, 3 };

         for (int i = 0; i < coinTypes.length; i++)
         {
            Coin c = coinTypes[i];
            int change = myRegister.giveChange(c);
            System.out.println(c.getName() + ": " + change);
            System.out.println("Expected: " + expected[i]);
         }
      }
   }
```

★ **Exercise P8.3.** Real cash registers can handle both bills and coins. Design a single class that expresses the commonality of these concepts. Redesign the `CashRegister` class and provide a method for entering payments that are described by your class. Your primary challenge is to come up with a good name for this class.

Your tester class should be called `CashRegisterTester`.

★ **Exercise P8.4.** Enhance the `BankAccount` class by adding preconditions for the constructor and the deposit method that require the `amount` parameter to be at least zero, and a precondition for the `withdraw` method that requires `amount` to be a value between 0 and the current balance. Use assertions to test the preconditions.

★ ★ **Exercise P8.5.** Write static methods

```
public static double sphereVolume(double r)
```


```
public static double sphereSurface(double r)
```


```
public static double cylinderVolume(double r, double h)
```


```
public static double cylinderSurface(double r, double h)
```


```
public static double coneVolume(double r, double h)
```


```
public static double coneSurface(double r, double h)
```


that compute the volume and surface area of a sphere with radius `r`, a cylinder with circular base with radius `r` and height `h`, and a cone with circular base with radius `r` and height `h`. Place them into a class `Geometry`. Then write a program that prompts the user for the values of `r` and `h`, calls the six methods, and prints the results.

Here is a sample program run:

```
Please enter the radius:
5
Please enter the height:
10
The volume of the sphere is: 523.5987755982989
The surface area of the sphere is: 314.1592653589793
The volume of the cylinder is: 785.3981633974483
The surface area of the cylinder is: 471.23889803846896
```

```
The volume of the cone is: 261.79938779914943
The surface area of the cone is: 235.61944901923448
```

Your main class should be called `GeometryCalculator`.

★★ **Exercise P8.6.** Solve Exercise P8.5 by implementing classes `Sphere`, `Cylinder`, and `Cone`. Which approach is more object-oriented?

Here is a sample program run:

```
Please enter the radius:
5
Please enter the height:
10
The volume of the sphere is: 523.5987755982989
The surface area of the sphere is: 314.1592653589793
The volume of the cylinder is: 785.3981633974483
The surface area of the cylinder is: 471.23889803846896
The volume of the cone is: 261.79938779914943
The surface area of the cone is: 235.61944901923448
```

Your main class should be called `GeometryCalculator`.

★★ **Exercise P8.7.** Write methods

```
public static double perimeter(Ellipse2D.Double e);
public static double area(Ellipse2D.Double e);
```

that compute the area and the perimeter of the ellipse e. Add these methods to a class Geometry. The challenging part of this assignment is to find and implement an accurate formula for the perimeter. Why does it make sense to use a static method in this case?

Use the following class as your tester class:

```
import java.awt.geom.Ellipse2D;
/**
   This is a tester for the ellipse geometry methods.
*/
public class EllipseTester
{
   public static void main(String[] args)
   {
      Ellipse2D.Double e = new Ellipse2D.Double(100, 100, 200, 100);
      System.out.println("Area: " + Geometry.area(e));
      System.out.println("Expected: 15707.963267948966192313 22");
      System.out.println("Perimeter: " + Geometry.perimeter(e));
      System.out.println("Expected: 484.42241102738376");
      // from http://www.jgiesen.de/kepler/arc/ellipseArc2.html
   }
}
```

★★ **Exercise P8.8.** Write methods

```
public static double angle(Point2D.Double p, Point2D.Double q)
public static double slope(Point2D.Double p, Point2D.Double q)
```

that compute the angle between the x-axis and the line joining two points, measured in degrees, and the slope of that line. Add the methods to the class `Geometry`. Supply suitable preconditions. Why does it make sense to use a static method in this case?

Use the following class as your tester class:

```
import java.awt.geom.Point2D;

/**
   This program tests the methods to compute the slope and
   angle of a line.
*/
public class LineTester
{
   public static void main(String[] args)
   {
      Point2D.Double p = new Point2D.Double(1, 1);
      Point2D.Double q = new Point2D.Double(3, 0);

      System.out.println("Slope: " + Geometry.slope(p, q));
      System.out.println("Expected: -0.5");

      Point2D.Double r = new Point2D.Double(3, -1);

      System.out.println("Angle: " + Geometry.angle(p, r));
      System.out.println("Expected: -45");

   }
}
```

★ ★ ★ **Exercise P8.9.** Write methods

```
public static boolean isInside(Point2D.Double p, Ellipse2D.Double e)
public static boolean isOnBoundary(Point2D.Double p, Ellipse2D.Double e)
```

that test whether a point is inside or on the boundary of an ellipse. Add the methods to the class `Geometry`.

Use the following class as your tester class:

```
import java.util.Scanner;
import java.awt.geom.Point2D;
import java.awt.geom.Ellipse2D;

/**
   This is a test driver for the ellipse containment methods.
*/
public class EllipseTester
{
   public static void main(String[] args)
   {
      Ellipse2D.Double e = new Ellipse2D.Double(100, 100, 200, 300);
      Point2D.Double p = new Point2D.Double(200, 100);
      System.out.pri ntln(Geometry.i sOnBoundary(p, e));
      System.out.println("Expected: true");
      Point2D.Double q = new Point2D.Double(0, 0);
      System.out.println(Geometry.isInside(q, e));
      System.out.println("Expected: false");
   }
}
```

★ **Exercise P8.10.** Write a method

```
public static int readInt(
      Scanner in, String prompt, String error, int min, int max)
```

that displays the prompt string, reads an integer, and tests whether it is between the minimum and maximum. If not, print an error message and repeat reading the input. Add the method to a class `Input`.

Use the following class as your main class:

```
import java. util.Scanner;

/**
   This program prints how old you'll be next year.
*/
public class AgePrinter
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      int age = Input.readInt(in, "Please enter your age",
         "Illegal Input--try again", 1, 150);
      System.out.println("Next year, you'll be " + (age + 1));
   }
}
```

★ ★ **Exercise P8.11.** Consider the following algorithm for computing x^n for an integer n. If $n < 0$, x^n is $1/x^{-n}$. If n is positive and even, then $x^n = (x^{n/2})^2$. If n is positive and odd, then $x^n = x^{n-1}$ • x. Implement a static method `double intPower(double x, int n)` that uses this algorithm. Add it to a class called `Numeric`.

Use the following class as your tester class:

```
/**
   This is a test driver for the intPower method.
*/
public class PowerTester
{
   public static void main(String[] args)
   {
      System.out.println(Numeric.intPower(0.1, 12));
      System.out.println("Expected: " + 1E-12);
      System.out.println(Numeric.intPower(2, 10));
      System.out.println("Expected: 1024");
      System.out.println(Numeric.intPower(-1, 1000));
      System.out.println("Expected: 1");
   }
}
```

★ ★ **Exercise P8.12.** Improve the `Needle` class of Chapter 6. Turn the `generator` field into a static field so that all needles share a single random number generator.

★ ★ **Exercise P8.13.** Implement a `Coin` and `CashRegister` class as described in Exercise P8.1. Place the classes into a package called `money`. Keep the `CashRegisterTester` class in the default package.

★ **Exercise P8.14.** Place a `BankAccount` class in a package whose name is derived from your e-mail address, as described in Section 8.9. Keep the `BankAccountTester` class in the default package.

★ ★ **Exercise P8.15.** Provide a JUnit test class `BankTest` with three test methods, each of which tests a different method of the `Bank` class in Chapter 7.

★ ★ **Exercise P8.16.** Provide JUnit test class `TaxReturnTest` with three test methods that test different tax situations for the Tax class in Chapter 5.

★ **Exercise P8.17.** Write methods

```
public static void drawH(Graphics2D g2, Point2D.Double p);
```

•

- public static void drawE(Graphics2D g2, Point2D.Double p);
- public static void drawL(Graphics2D g2, Point2D.Double p);
- public static void drawO(Graphics2D g2, Point2D.Double p);

that show the letters H, E, L, O on the graphics window, where the point p is the top-left corner of the letter. Then call the methods to draw the words "HELLO" and "HOLE" on the graphics display. Draw lines and ellipses. Do not use the `drawString` method. Do not use `System.out`.

Here is a sample program output:

Use the following class as your main class:

```
import javax.swing.JFrame;
/**
   This program shows the message "HELLO".
*/
public class HelloViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 200;
      final int FRAME_HEIGHT = 150;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("HelloViewer");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      LettersComponent component = new LettersComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

Use the following class in your solution:

```
import javax.swing.JComponent;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Point2D;

public class LettersComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      final int WIDTH = 30;
      final int HEIGHT = 50;
```

```
      final int SPACING = 10;

      Graphics2D g2 = (Graphics2D) g;

      Rectangle bounds = new Rectangle(0, 0, WIDTH, HEIGHT);

      Letters.drawH(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawE(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawL(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawL(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawO(g2, bounds);
      bounds = new Rectangle(0, HEIGHT + SPACING, WIDTH, HEIGHT);

      Letters.drawH(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawO(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawL(g2, bounds);
      bounds.translate(WIDTH + SPACING, 0);

      Letters.drawE(g2, bounds);
   }
}
```

★ ★ **Exercise P8.18.** Repeat Exercise P8.15 by designing classes `LetterH`, `LetterE`, `LetterL`, and `LetterO`, each with a constructor that takes a Point2D.Double parameter (the top-left corner) and a method `draw(Graphics2D g2)`.Which solution is more object-oriented?

Here is a sample program output:

Use the following class as your main class:

```
import javax.swing.JFrame;

public class HelloViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 200;
      final int FRAME_HEIGHT = 150;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("HelloViewer");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        LettersComponent component = new LettersComponent();
        frame.add(component);

        frame.setVisible(true);
    }
```