Design

Our rather crude estimate for π is 3.14. Obviously, more accurate estimates yield more accurate calculations of the area. Later we see how Java itself can provide an excellent estimate. Following is the pseudocode for the count angels program:

```
read radius
read angelSpace
read overlapFactor
area = 3.14 * radius * radius
nonOverlapSpace = angelSpace * (1.0 - overlapFactor)
numberAngels = area / nonOverlapSpace
print numberAngels
```

Implementation

The following code is a straightforward translation of the pseudocode into Java. Comments are included.

```
/*Case study 2: count the angels
Count the number of angels that can dance on the head of a pin.
1. The user inputs are
       The radius of the pinhead
       The space occupied by an angel
       The allowed overlap between angels subject to the restriction
       that no space can simultaneously be occupied by more than two
2. The program computes
       The area of the pinhead based on its radius
       The amount of nonoverlapping space required by an angel
       The number of angels based on the preceding two values
3. The program ends by printing the number of angels.
*/

import java.util.Scanner;

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

       Scanner reader = new Scanner(System.in);

       double radius;          //Radius of the pinhead in millimeters

       double angelSpace;      //Space occupied by an angel
                               //in square micrometers
       double overlapFactor;   //Allowed overlap between angels from 0 to 1
       double area;            //Area of the pinhead in square millimeters
       double nonOverlapSpace; //Nonoverlapping space required by an angel
       double numberAngels;    //Number of angels that can dance on the
                               //pinhead

      //Get user inputs
      System.out.print ("Enter the radius in millimeters: ");
      radius = reader.nextDouble();
      System.out.print
```

```
            ("Enter the space occupied by an angel in square micrometers: ");
         angelSpace = reader.nextDouble();
         System.out.print ("Enter the overlap factor: ");
         overlapFactor = reader.nextDouble();

         //Perform calculations
         area = 3.14 * radius * radius;
         nonOverlapSpace = angelSpace * (1.0 - overlapFactor);
         numberAngels = area / nonOverlapSpace;

         //Print results
         System.out.print ("The number of angels = " + numberAngels);
      }
   }
```

Discussion

So what were the mysterious errors we mentioned and what is their general significance? There were three errors, two during analysis and one during design.

First Analysis Error

During analysis we did not consider the shape of the region occupied by an angel, overlapping or otherwise. To appreciate the significance of our oversight, consider the problem of placing as many pennies as possible on a plate without overlap. Because there are gaps between the pennies, the answer is not obtained by dividing the area of the plate by the area of a penny. Even if two pennies are allowed to overlap by some amount, there are still gaps. Thus our solution is correct only if angels can mold their shapes to eliminate all empty spaces. Unfortunately, we did not think of asking the theologians about this.

Second Analysis Error

Let us now simplify the problem and suppose that angels pack onto the pinhead without leaving empty spaces. Now the space occupied by two overlapping angels equals the space each would occupy alone minus the amount by which they overlap or

```
space for two overlapping angels
      = 2 * space occupied by an angel -
          space occupied by an angel * overlap factor
      = 2 * space occupied by an angel * (1.0 - overlap factor / 2)
```

Thus,

```
space for one angel with overlap = space occupied by an angel *
  (1.0 - overlap factor / 2)
```

and

```
number of angels on pinhead = area of pinhead / space for one
  angel with overlap
```

Well, we certainly got that wrong the first time.

Design Error

The radius of the pin is given in millimeters and the space requirements of an angel are given in square micrometers. Our calculations need to take this difference in units into account. We leave the actual correction as an exercise.

Conclusions

There are three lessons to draw from all this. First, the people who write programs usually are not the ones most familiar with the problem domain. Consequently, many programs fail to solve problems correctly either because they completely ignore important factors or because they treat factors incorrectly. Second, careful analysis and design are essential and demand careful thought. As you can see, the errors had nothing to do with programming per se, and they would have occurred even if we were solving the problem with paper and pencil. And by the way, before writing a program to solve a problem, we definitely need to know how to do it correctly by hand. We are not going to make a practice of making analysis and design errors, and we did so just this once in order to make a point. Third, just because computers perform complex calculations at lightning speed does not mean we should have unquestioning confidence in their outputs.

3.7 *Graphics and GUIs: Drawing Shapes and Text*

The GUI programs in Section 2.7 displayed one or more flat-colored rectangular areas. Now it is time to learn how to fill these areas with other objects, such as geometric shapes and text.

Extra Challenge

This Graphics and GUIs section gives you the opportunity to explore concepts and programming techniques required to develop modern graphics applications and graphical user interfaces. This material is not required in order to proceed with the other chapters of the book.

Defining a Specialized Panel Class

An application window in GUI programs has a clearly defined set of responsibilities. A window establishes its initial size, decides what it will do when the user resizes or closes it, and creates and lays out the panels that appear within it. Before we create and display other objects within a panel, we have to ask which will be responsible for them, the application window or the panel in which they appear? Because there might be many panels and many objects in each panel, it would be a good idea to assign at least some of the responsibilities for managing these objects to the panels themselves. We use two principles of any good organization: Divide the labor and Delegate responsibility.

A window does just what it did before: create a panel with a given background color. But a panel now has the additional responsibility of setting its own background color and displaying geometric shapes and text. To provide a panel with these additional capabilities, we define a new type of panel by extending the `JPanel` class. Our new class, called `ColorPanel`, has all of the behavior of a `JPanel`. But a `ColorPanel` also knows how to draw specific shapes and instantiate itself with a given background color. For example, instead of the code

```
JPanel panel = new JPanel();
panel.setBackround(Color.white);
```

an application window now uses the code

```
ColorPanel panel = new ColorPanel(Color.white);
```

to create a panel with a given background color. The following program sets up an application window for all of the examples discussed in this section.

```
// Main application window for Chapter 3 graphics examples

import javax.swing.*;
import java.awt.*;

public class GUIWindow {

   public static void main(String[] args){
      JFrame theGUI = new JFrame();
      theGUI.setTitle("GUI Program");
      theGUI.setSize(300, 200);
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      ColorPanel panel = new ColorPanel(Color.white);
      Container pane = theGUI.getContentPane();
      pane.add(panel);
      theGUI.setVisible(true);
    }
  }
```

The class `ColorPanel` in each example extends `JPanel` and includes a constructor that expects a `Color` parameter. The constructor runs when the panel is instantiated and sets its background color. Here is the code for this minimal behavior:

```
// Example 3.5: an empty colored panel

import javax.swing.*;
import java.awt.*;

public class ColorPanel extends JPanel{

   public ColorPanel(Color backColor){
      setBackground(backColor);
   }
}
```

All we have done thus far is substitute a new class, `ColorPanel`, for the `JPanel` class used in the examples of Section 2.7. Those example programs would have the same behavior if they used the new class. The only difference is that the code for the application windows would be slightly simplified.

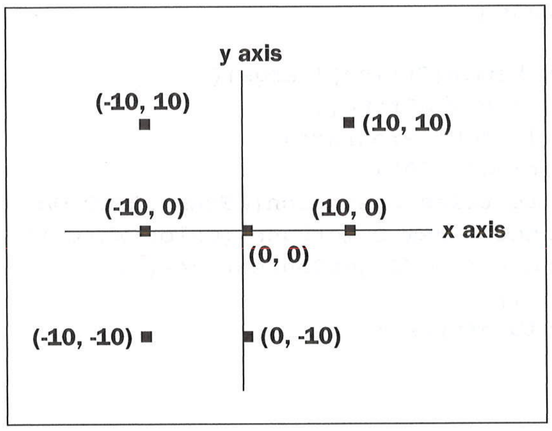
When you're working with more than one programmer-defined class, the source code for the classes is usually maintained in separate files and edited and compiled separately. An easy way to compile all of the classes in your current working directory is to use the command `javac *.java` at the command line prompt. The `java` command to run a program is used as before, with the byte code file that contains the `main` method.

Coordinate Systems

Underlying every graphics application is a ***coordinate system***. Positions in this system are specified in terms of points. Points in a two-dimensional system have *x* and *y* coordinates. For example, the point (10, 30) has an *x* coordinate of 10 and a *y* coordinate of 30.

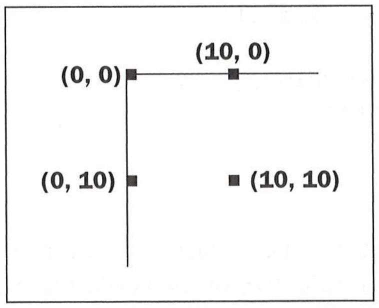
The *x* and *y* coordinates of a point express its position relative to the system's ***origin*** at (0, 0). Figure 3-6 presents some examples of points in the familiar Cartesian coordinate system. In this system, two perpendicular lines define an *x* axis and a *y* axis. The point of intersection is labeled (0, 0). Increasing values of *x* are to the right and increasing values of *y* are upward.

FIGURE 3-6
A Cartesian coordinate system



In Java and most other programming languages, the coordinate system is oriented as shown in Figure 3-7. Note that the only quadrant shown is the one that defines the coordinates of the computer's screen. In the positive direction, it extends downward and to the right from the point (0, 0) in the upper-left corner. The other three quadrants exist, but the points in them never appear on the screen. This is called a ***screen coordinate system***.

FIGURE 3-7
Orientation of Java's coordinate system



In a window-based application, each window has a coordinate system whose origin is located at the upper-left outside corner of the window. Each integer point in this coordinate system, extending from the origin to the window's lower-right corner, locates the position of a pixel, or picture element, in the window. By an integer point, we mean a point both of whose coordinates are integers. Each panel also has its own coordinate system that is similar in form to the window's coordinate system. In the discussion that follows, we assume that we are drawing geometric shapes and text in panels.

The Graphics Class

The package `java.awt` provides a `Graphics` class for drawing in a panel. A panel maintains an instance of this class, called a ***graphics context***, so that the program can access and modify the panel's bitmap. The program sends messages to the graphics context to perform all graphics operations. Hereafter, we refer to the graphics context using the variable name `g`. Some commonly used `Graphics` drawing methods are listed in Table 3-9. The table also shows the results of running these methods in a window that has a single panel.

TABLE 3-9
Common methods in the `Graphics` class

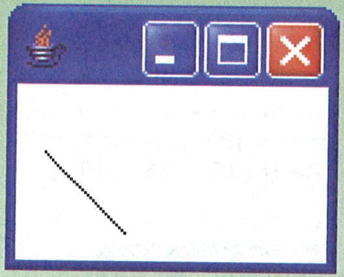
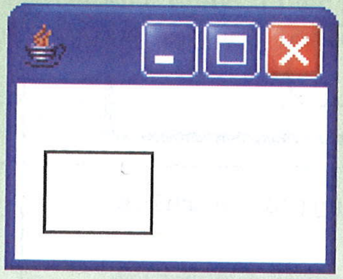
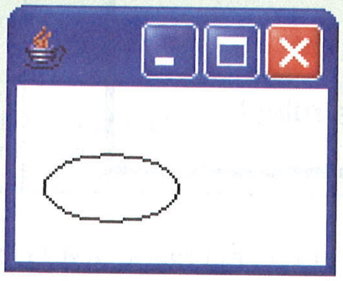
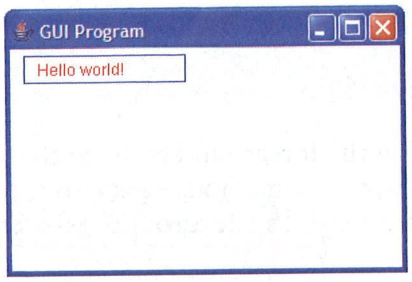
GRAPHICS METHOD	EXAMPLE CALL AND OUTPUT	WHAT IT DOES
`drawLine( int x1, int y1, int x2, int y2)`	`g.drawLine(10, 25, 40, 55)` 	Draws a line from point (x1, y1) to (x2, y2).
`drawRect( int x, int y, int width, int height)`	`g.drawRect(10, 25, 40, 30)` 	Draws a rectangle whose upper-left corner is (x, y) and whose dimensions are the specified width and height.
`drawOval( int x, int y, int width, int height)`	`g.drawOval(10, 25, 50, 25)` 	Draws an oval that fits within a rectangle whose origin (upper-left corner) is (x, y) and whose dimensions are the specified width and height. To draw a circle, make the width and height equal.

FIGURE 3-8
Displaying a shape and text in a panel



Note that the `paintComponent` method first calls the same method in the superclass, using the reserved word `super`. The reason is that the method in the superclass paints the background of the panel. This effectively clears any images in the panel before they are redrawn.

Finding the Width and Height of a Panel

Occasionally, it is useful to know the width and height of a panel. For instance, one might want to center an image in a panel and keep the image centered when the user resizes the window. The methods `getWidth()` and `getHeight()` return the current width and height of a panel, respectively. If these methods are called before the window is opened, they return a default value of 0. Our next `ColorPanel` example maintains the shapes of the previous example near the center of the panel:

```
// Example 3.7: A colored panel containing a red text
// message in a blue rectangle, centered in the panel

import javax.swing.*;
import java.awt.*;

public class ColorPanel extends JPanel{

   public ColorPanel(Color backColor){
      setBackground(backColor);
   }

   public void paintComponent(Graphics g){
      super.paintComponent(g);
      int x = getWidth() / 2 - 60;
      int y = getHeight() / 2;
      g.setColor(Color.blue);
      g.drawRect(x, y, 120, 20);
      g.setColor(Color.red);
      g.drawString("Hello world!", x + 10, y + 15);
   }
}
```

Text Properties and the `Font` Class

In the context of a bitmapped display, text is drawn like any other image. A text image has several properties, as shown in Table 3-10. These are set by adjusting the color and font properties of the graphics context in which the text is drawn.

10
perties

XT PROPERTY	EXAMPLE
olor	Red, green, blue, white, black, etc.
Font style	Plain, **bold**, *italic*
Font size	10 point, 12 point, etc.
Font name	Courier, Times New Roman, etc.

An object of class `Font` has three basic properties: a name, a style, and a size. The following code creates one `Font` object with the properties **Courier bold 12** and another with the properties ***Arial bold italic 10***:

```
Font courierBold12      = new Font("Courier", Font.BOLD, 12);
Font arialBoldItalic10 = new Font("Arial", Font.BOLD + Font.ITALIC, 10);
```

The `Font` constants `PLAIN`, `BOLD`, and `ITALIC` define the font styles. The font size is an integer representing the number of points, where one point equals 1/72 of an inch. The available font names depend on your particular computer platform.

Our final program example modifies the previous one so that the text message is displayed in Courier bold 14 font.

```
// Example 3.8: A colored panel containing a red text
// message in a blue rectangle
// Text font is Courier bold 14

import javax.swing.*;
import java.awt.*;

public class ColorPanel extends JPanel{

   public ColorPanel(Color backColor){
      setBackground(backColor);
   }

   public void paintComponent(Graphics g){
      super.paintComponent(g);
      int x = getWidth() / 2 - 60;
      int y = getHeight() / 2;
      g.setColor(Color.blue);
      g.drawRect(x, y, 120, 20);
      g.setColor(Color.red);
      Font font = new Font("Courier", Font.BOLD, 14);
      g.setFont(font);
      g.drawString("Hello world!", x + 10, y + 15);
   }
}
```

EXERCISE 3.7

1. Write the code segments that would draw the following objects in the graphics context `g`:
   a. A filled rectangle with corner point (45, 20) and size 100 by 50
   b. A line segment with end points (20, 20) and (100, 100)
   c. A circle with center point (100, 100) and radius 50
   d. A triangle with vertices (100, 100), (50, 50), and (200, 200)
2. Describe the design of a program that displays a filled blue rectangle on a red background.
3. How does one compute the center point of a panel?
4. List the three properties of a text font.

SUMMARY

In this chapter, you learned:

- Java programs use the `int` data type for whole numbers (integers) and `double` for floating-point numbers (numbers with decimals).
- Java variable and method names consist of a letter followed by additional letters or digits. Java keywords cannot be used as names.
- Final variables behave as constants; their values cannot change after they are declared.
- Arithmetic expressions are evaluated according to precedence. Some expressions yield different results for integer and floating-point operands.
- Strings may be concatenated to form a new string.
- The compiler catches syntax errors. The JVM catches run-time errors. Logic errors, if they are caught, are detected by the programmer or user of the program at runtime.
- A useful way to find and remove logic errors is to insert debugging output statements to view the values of variables.
- Java uses a screen coordinate system to locate the positions of pixels in a window or panel. The origin of this system is in the upper-left corner or the drawing area, and the *x* and *y* axes increase to the right and downward, respectively.
- The programmer can modify the color with which images are drawn and the properties of text fonts for a given graphics object.

VOCABULARY *Review*

Define the following terms:

Arithmetic expression
Comments
Coordinate system
Exception
Graphics context
Literal
Logic error
Origin
Package
Pseudocode
Reserved words
Run-time error
Screen coordinate system
Semantics
Syntax
Virus

REVIEW *Questions*

WRITTEN QUESTIONS

Write a brief answer to the following questions.

1. Write a pseudocode algorithm that determines the batting average of a baseball player. *Hint*: To compute a batting average, divide number of hits by number of at-bats. Batting averages have three decimal places.

2. Give examples of an integer literal, a floating-point literal, and a string literal.

3. Declare variables to represent a person's name, age, and hourly wage.

4. Why must care be taken to order the operators in an arithmetic expression?

5. Is it possible to assign a value of type `int` to a variable of type `double`? Why or why not?

6. State which of the following are valid Java identifiers. For those that are not valid, explain why.
   A. length

   B. import

   C. 6months

   D. hello-and-goodbye

   E. HERE_AND_THERE

FILL IN THE BLANK

Complete the following sentences by writing the correct word or words in the blanks provided.

1. In mixed-mode arithmetic with operand types `int` and `double`, the result type is always ________.
2. A method's name, parameters, and return type are also known as its ________.
3. The operation that joins two strings together is called ________.
4. End-of-line comments begin with the symbol ________.
5. A quotient results when the ________ operator is used with two operands of type ________.

PROJECTS

PROJECT 3-1

The surface area of a cube can be known if we know the length of an edge. Write a program that takes the length of an edge (an integer) as input and prints the cube's surface area as output. (*Remember*: analyze, design, implement, and test.)

PROJECT 3-2

Write a program that takes the radius of a sphere (a double) as input and outputs the sphere's diameter, circumference, surface area, and volume.

PROJECT 3-3

The kinetic energy of a moving object is given by the formula $KE=(1/2)mv^2$, where m is the object's mass and v is its velocity. Modify the program of Chapter 2, Project 2-5, so that it prints the object's kinetic energy as well as its momentum.

PROJECT 3-4

An employee's total weekly pay equals the hourly wage multiplied by the total number of regular hours plus any overtime pay. Overtime pay equals the total overtime hours multiplied by 1.5 times the hourly wage. Write a program that takes as inputs the hourly wage, total regular hours, and total overtime hours and displays an employee's total weekly pay.

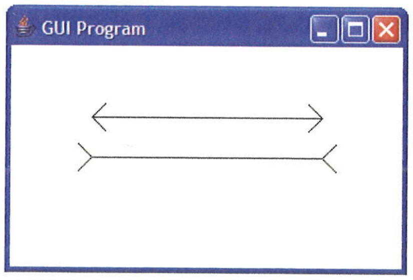
PROJECT 3-5

Modify the program of Project 3-4 so that it prompts the user for the regular and overtime hours of each of five working days.

PROJECT 3-6

The Müller-Lyer illusion is caused by an image that consists of two parallel line segments. One line segment looks like an arrow with two heads, and the other line segment looks like an arrow with two tails. Although the line segments are of exactly the same length, they appear to be unequal (see Figure 3-9). Write a graphics program that illustrates this illusion.

FIGURE 3-9
The Müller-Lyer Illusion



PROJECT 3-7

Write a graphics program that displays the coordinates of the center point of a panel in the form (*x*, *y*). This information should be displayed at the panel's center point and be automatically updated when the panel is resized.

CRITICAL *Thinking*

During the summer before the academic year, the registrar's office must enter new data for incoming freshmen. Design and implement a program that prompts the user for the following inputs:

Last name

First name

Class year (an integer)

Campus phone

After all the inputs are taken, the program should echo them as output.