Summary of Learning Objectives

Declare objects, classes, and methods.

- Objects are entities in your program that you manipulate by calling methods.
- A method is a sequence of instructions that accesses the data of an object.
- A class declares the methods that you can apply to its objects.
- The return value of a method is a result that the method has computed for use by the code that called it.
- A parameter is an input to a method.
- The implicit parameter of a method call is the object on which the method is invoked. All other parameters are explicit parameters.

Write variable declarations in Java.

- You use variables to store values that you want to use at a later time. A variable has a type, a name, and a value.
- Identifiers for variables, methods, and classes are composed of letters, digits, and the underscore character.
- By convention, variable names should start with a lowercase letter.
- The `int` type denotes integers. The `double` type denotes floating-point numbers that can have fractional parts.
- Use the assignment operator (=) to change the value of a variable.
- Numbers and variables can be combined by arithmetic operators such as +, -, and *.
- All variables must be initialized before you access them.

Use constructors to construct new objects.

- Use the `new` operator, followed by a class name and parameters, to construct new objects.

Use the API documentation for finding method descriptions and packages.

- The API (Application Programming Interface) documentation lists the classes and methods of the Java library.
- Java classes are grouped into packages. Use the `import` statement to use classes that are declared in other packages.

Write programs that test behavior of methods.

- A test program verifies that methods behave as expected.
- Determining the expected result in advance is an important part of testing.

Understand instance variables and the methods that access them.

- An object's instance variables store the data required for executing its methods.
- Each object of a class has its own set of instance variables.
- Private instance variables can only be accessed by methods of the same class.
- Encapsulation is the process of hiding implementation details and providing methods for data access.
- Encapsulation allows a programmer to use a class without having to know its implementation.

Write method and constructor headers that describe the public interface of a class.

- The public interface of a class specifies what you can do with its objects.
- In order to implement a class, you first need to know which methods are required.
- An accessor method does not change the internal data of its implicit parameter. A mutator method changes the data.
- In a method header, you specify the return type, method name, and the types and names of the parameters.
- Constructors set the initial data for objects. The constructor name is always the same as the class name.
- Use documentation comments to describe the classes and public methods of your programs.

Provide the private implementation of a class.

- The private implementation of a class consists of instance variables, and the bodies of constructors and methods.

Write tests that verify that a class works correctly.

- A unit test verifies that a class works correctly in isolation, outside a complete program.

Compare lifetime and initialization of instance, local, and parameter variables.

- Local variables are declared in the body of a method.
- When a method exits, its local variables are removed.
- Instance variables are initialized to a default value, but you must initialize local variables.

Describe how multiple object references can refer to the same object.

- An object reference describes the location of an object.
- Multiple object variables can contain references to the same object.
- Number variables store numbers. Object variables store references.

Recognize the use of the implicit parameter in method declarations.

- Use of an instance variable name in a method denotes the instance variable of the implicit parameter.
- The `this` reference denotes the implicit parameter.
- A method call without an implicit parameter is applied to the same object.

Write programs that display drawings in frame windows.

- To show a frame, construct a `JFrame` object, set its size, and make it visible.
- In order to display a drawing in a frame, declare a class that extends the `JComponent` class.
- Place drawing instructions inside the `paintComponent` method. That method is called whenever the component needs to be repainted.
- Use a cast to recover the `Graphics2D` object from the `Graphics` parameter of the `paintComponent` method.

Use the Java API for drawing simple figures.

- The `Ellipse2D.Double` and `Line2D.Double` classes describe graphical shapes.
- The `drawString` method draws a string, starting at its basepoint.
- When you set a new color in the graphics context, it is used for subsequent drawing operations.

Implement classes that draw graphical shapes.

- It is a good idea to make a class for any part of a drawing that can occur more than once.
- To figure out how to draw a complex shape, make a sketch on graph paper.

Classes, Objects, and Methods Introduced in this Chapter

```
java.awt.Color
java.awt.Component
   getHeight
   getWidth
   setSize
   setVisible
java.awt.Frame
   setTitle
java.awt.geom.Ellipse2D.Double
java.awt.geom.Line2D.Double
java.awt.geom.Point2D.Double
java.awt.Graphics
   setColor
java.awt.Graphics2D
   draw
   drawString
   fill
java.awt.Rectangle
   getX
   getY
   getHeight
   getWidth
   setSize
   translate
java.lang.String
   length
   replace
   toLowerCase
   toUpperCase
javax.swing.JComponent
   paintComponent
javax.swing.JFrame
   setDefaultCloseOperation
```

Media Resources

www.wiley.com/go/global/horstmann

- ***Worked Example*** How Many Days Have You Been Alive?
- ***Worked Example*** Working with Pictures
- ***Worked Example*** Making a Simple Menu
- Lab Exercises
- ***Animation*** Variable Initialization and Assignment
- ***Animation*** Lifetime of Variables
- ***Animation*** Object References
- Practice Quiz
- Code Completion Exercises

Review Exercises

★ **R2.1** Explain the difference between an object and an object reference.

★ **R2.2** Explain the difference between an object and an object variable.

★ **R2.3** Explain the difference between an object and a class.

★★ **R2.4** Give the Java code for constructing an *object* of class `Rectangle`, and for declaring an *object variable* of class `Rectangle`.

★★ **R2.5** Give Java code for objects with the following descriptions:

a. A rectangle with center (100, 100) and all side lengths equal to 50

b. A string with the contents "Hello, Dave"

Create objects, not object variables.

★★ **R2.6** Repeat Exercise R2.5, but now declare object variables that are initialized with the required objects.

★★ **R2.7** Write a Java statement to initialize a variable `square` with a rectangle object whose top-left corner is (10, 20) and whose sides all have length 40. Then write a statement that replaces `square` with a rectangle of the same size and top left corner (20, 20).

★★ **R2.8** Write Java statements that initialize two variables `square1` and `square2` to refer to the same square with center (20, 20) and side length 40.

★★ **R2.9** Find the errors in the following statements:

a.
```
Rectangle r = (5, 10, 15, 20);
```

b.
```
double width = Rectangle(5, 10, 15, 20).getWidth();
```

c.
```
Rectangle r;
r.translate(15, 25);
```

d.
```
r = new Rectangle();
r.translate("far, far away!");
```

★ **R2.10** Name two accessor methods and two mutator methods of the `Rectangle` class.

★★ **R2.11** Look into the API documentation of the `Rectangle` class and locate the method

```
void add(int newx, int newy)
```

Read through the method documentation. Then determine the result of the following statements:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
box.add(0, 0);
```

If you are not sure, write a small test program.

★ **R2.12** Consider a class `Grade` that represents a letter grade, such as A+ or B. Give two choices of instance variables that can be used for implementing the `Grade` class.

★★ **R2.13** Consider a class `Time` that represents a point in time, such as 9 A.M. or 3:30 P.M. Give two different sets of instance variables that can be used for implementing the `Time` class.

★ **R2.14** Suppose the implementor of the `Time` class of Exercise R2.13 changes from one implementation strategy to another, keeping the public interface unchanged. What do the programmers who use the `Time` class need to do?

★★ **R2.15** You can read the `value` instance variable of the `Counter` class with the `getValue` accessor method. Should there be a `setValue` mutator method to change it? Explain why or why not.

★★ **R2.16** Why does the `BankAccount` class not have a `reset` method?

★ **R2.17** What happens in our implementation of the `BankAccount` class when more money is withdrawn from the account than the current balance?

★★ **R2.18** What does the following method do? Give an example of how you can call the method.

```
public class BankAccount
{
   public void mystery(BankAccount that, double amount)
   {
      this.balance = this.balance - amount;
      that.balance = that.balance + amount;
   }
   . . . // Other bank account methods
}
```

★★ **R2.19** Suppose you want to implement a class `TimeDepositAccount`. A time deposit account has a fixed interest rate that should be set in the constructor, together with the initial balance. Provide a method to get the current balance. Provide a method to add the earned interest to the account. This method should have no parameters because the interest rate is already known. It should have no return value because you already provided a method for obtaining the current balance. It is not possible to deposit additional funds into this account. Provide a `withdraw` method that removes the entire balance. Partial withdrawals are not allowed.

★ **R2.20** Consider the following implementation of a class `Square`:

```
public class Square
{
   private int sideLength;
   private int area; // Not a good idea

   public Square(int length)
   {
      sideLength = length;
   }

   public int getArea()
   {
      area = sideLength * sideLength;
      return area;
   }
}
```

Why is it not a good idea to introduce an instance variable for the area? Rewrite the class so that `area` is a local variable.

★★ **R2.21** Consider the following implementation of a class `Square`:

```
public class Square
{
   private int sideLength;
   private int area;

   public Square(int initialLength)
   {
      sideLength = initialLength;
      area = sideLength * sideLength;
   }

   public int getArea() { return area; }
   public void grow() { sideLength = 2 * sideLength(); }
}
```

What error does this class have? How would you fix it?

★★T **R2.22** Provide a unit test class for the `Counter` class in Section 2.6.

★★T **R2.23** Read Exercise P2.17, but do not implement the `Car` class yet. Write a tester class that tests a scenario in which gas is added to the car, the car is driven, more gas is added, and the car is driven again. Print the actual and expected amount of gas in the tank.

★G **R2.24** What is the difference between a console application and a graphical application?

★★G **R2.25** Who calls the `paintComponent` method of a component? When does the call to the `paintComponent` method occur?

★★G **R2.26** Why does the parameter of the `paintComponent` method have type `Graphics` and not `Graphics2D`?

★★G **R2.27** Why are separate viewer and component classes used for graphical programs?

★★G **R2.28** Suppose you want to extend the car viewer program in Section 2.16 to show a suburban scene, with several cars and houses. Which classes do you need?

★★★G **R2.29** Explain why the calls to the `getWidth` and `getHeight` methods in the `CarComponent` class have no explicit parameter.

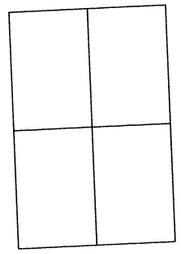
★★G **R2.30** How would you modify the `Car` class in order to show cars of varying sizes?

Programming Exercises

★T **P2.1** Write an `AreaTester` program that constructs a `Rectangle` object and then computes and prints its area. Use the `getWidth` and `getHeight` methods. Also print the expected answer.

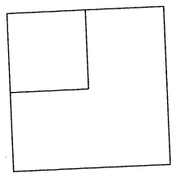
★T **P2.2** Write a `PerimeterTester` program that constructs a `Rectangle` object and then computes and prints its perimeter. Use the `getWidth` and `getHeight` methods. Also print the expected answer.

★★ **P2.3** Write a program called `FourRectanglePrinter` that constructs a `Rectangle` object, prints its location by calling `System.out.println(box)`, and then translates and prints it three more times, so that, if the rectangles were drawn, they would form one large rectangle:



Your program will not produce a drawing. It will simply print the locations of the four rectangles.

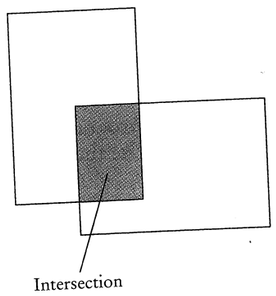
★★ **P2.4** Write a `GrowSquarePrinter` program that constructs a `Rectangle` object `square` representing a square with top-left corner (100, 100) and side length 50, prints its location by calling `System.out.println(square)`, applies the `translate` and `grow` methods and calls `System.out.println(square)` again. The calls to `translate` and `grow` should modify the square so that it has twice the size and the same top-left corner as the original. If the squares were drawn, they would look like this:



Your program will not produce a drawing. It will simply print the locations of square before and after calling the mutator methods.

Look up the description of the `grow` method in the API documentation.

★★★ **P2.5** The `intersection` method computes the *intersection* of two rectangles—that is, the rectangle that would be formed by two overlapping rectangles if they were drawn:



You call this method as follows:

```
Rectangle r3 = r1.intersection(r2);
```

Write a program `IntersectionPrinter` that constructs two rectangle objects, prints them as described in Exercise P2.3, and then prints the rectangle object that

describes the intersection. Then the program should print the result of the `intersection` method when the rectangles do not overlap. Add a comment to your program that explains how you can tell whether the resulting rectangle is empty.

★★★ **P2.6** In this exercise, you will explore a simple way of visualizing a `Rectangle` object. The `setBounds` method of the `JFrame` class moves a frame window to a given rectangle. Complete the following program to visually show the `translate` method of the `Rectangle` class:

```
import java.awt.Rectangle;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class TranslateDemo
{
   public static void main(String[] args)
   {
      // Construct a frame and show it
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

      // Your work goes here:
      // Construct a rectangle and set the frame bounds

      JOptionPane.showMessageDialog(frame, "Click OK to continue");

      // Your work goes here:
      // Move the rectangle and set the frame bounds again
   }
}
```

★★ **P2.7** In the Java library, a color is specified by its red, green, and blue components between 0 and 255 (see Table 2 on page 74). Write a program `BrighterDemo` that constructs a `Color` object with red, green, and blue values of 50, 100, and 150. Then apply the `brighter` method and print the red, green, and blue values of the resulting color. (You won't actually see the color—see the next exercise on how to display the color.)

★★ **P2.8** Repeat Exercise P2.7, but place your code into the following class. Then the color will be displayed.

```
import java.awt.Color;
import javax.swing.JFrame;

public class BrighterDemo
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();
      frame.setSize(200, 200);
      Color myColor = ...;
      frame.getContentPane().setBackground(myColor);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

★★ **P2.9** Repeat Exercise P2.7, but apply the `darker` method twice to the object `Color.RED`. Call your class `DarkerDemo`.

★★ **P2.10** The `Random` class implements a *random number generator*, which produces sequences of numbers that appear to be random. To generate random integers, you construct an object of the `Random` class, and then apply the `nextInt` method. For example, the call `generator.nextInt(6)` gives you a random number between 0 and 5.

Write a program `DieSimulator` that uses the `Random` class to simulate the cast of a die, printing a random number between 1 and 6 every time that the program is run.

★T **P2.11** Write a `BankAccountTester` class whose `main` method constructs a bank account, deposits $1,000, withdraws $500, withdraws another $400, and then prints the remaining balance. Also print the expected result.

★ **P2.12** Add a method

```
public void addInterest(double rate)
```

to the `BankAccount` class that adds interest at the given rate. For example, after the statements

```
BankAccount momsSavings = new BankAccount(1000);
momsSavings.addInterest(10); // 10 percent interest
```

the balance in `momsSavings` is $1,100. Also supply a `BankAccountTester` class that prints the actual and expected balance.

★★ **P2.13** Write a class `SavingsAccount` that is similar to the `BankAccount` class, except that it has an added instance variable `interest`. Supply a constructor that sets both the initial balance and the interest rate. Supply a method `addInterest` (with no explicit parameter) that adds interest to the account. Write a `SavingsAccountTester` class that constructs a savings account with an initial balance of $1,000 and an interest rate of 10 percent. Then apply the `addInterest` method and print the resulting balance. Also compute the expected result by hand and print it.

★★★ **P2.14** Add a feature to the `CashRegister` class for computing sales tax. The tax rate should be supplied when constructing a `CashRegister` object. Add `recordTaxablePurchase` and `getTotalTax` methods. (Amounts added with `recordPurchase` are not taxable.) The `giveChange` method should correctly reflect the sales tax that is charged on taxable items.

★★ **P2.15** After closing time, the store manager would like to know how much business was transacted during the day. Modify the `CashRegister` class to enable this functionality. Supply methods `getSalesTotal` and `getSalesCount` to get the total amount of all sales and the number of sales. Supply a method `reset` that resets any counters and totals so that the next day's sales start from zero.

★★ **P2.16** Implement a class `Employee`. An employee has a name (a string) and a salary (a `double`). Provide a constructor with two parameters

```
public Employee(String employeeName, double currentSalary)
```

and methods

```
public String getName()
public double getSalary()
public void raiseSalary(double byPercent)
```


These methods return the name and salary, and raise the employee's salary by a certain percentage. Sample usage:

```
Employee harry = new Employee("Morgan, Harry", 50000);
harry.raiseSalary(10); // Harry gets a 10 percent raise
```

Supply an `EmployeeTester` class that tests all methods.

★★ **P2.17** Implement a class `Car` with the following properties. A car has a certain fuel efficiency (measured in miles/gallon or liters/km—pick one) and a certain amount of fuel in the gas tank. The efficiency is specified in the constructor, and the initial fuel level is 0. Supply a method `drive` that simulates driving the car for a certain distance, reducing the amount of gasoline in the fuel tank. Also supply methods `getGasInTank`, returning the current amount of gasoline in the fuel tank, and `addGas`, to add gasoline to the fuel tank. Sample usage:

```
Car myHybrid = new Car(50); // 50 miles per gallon
myHybrid.addGas(20); // Tank 20 gallons
myHybrid.drive(100); // Drive 100 miles
double gasLeft = myHybrid.getGasInTank(); // Get gas remaining in tank
```

You may assume that the `drive` method is never called with a distance that consumes more than the available gas. Supply a `CarTester` class that tests all methods.

★★ **P2.18** Implement a class `Student`. For the purpose of this exercise, a student has a name and a total quiz score. Supply an appropriate constructor and methods `getName()`, `addQuiz(int score)`, `getTotalScore()`, and `getAverageScore()`. To compute the latter, you also need to store the *number of quizzes* that the student took.

Supply a `StudentTester` class that tests all methods.

★ **P2.19** Implement a class `Product`. A product has a name and a price, for example `new Product("Toaster", 29.95)`. Supply methods `getName`, `getPrice`, and `reducePrice`. Supply a program `ProductPrinter` that makes two products, prints the name and price, reduces their prices by $5.00, and then prints the prices again.

★★ **P2.20** Write a class `Bug` that models a bug moving along a horizontal line. The bug moves either to the right or left. Initially, the bug moves to the right, but it can turn to change its direction. In each move, its position changes by one unit in the current direction. Provide a constructor

```
public Bug(int initialPosition)
```

and methods

```
public void turn()
public void move()
public int getPosition()
```

Sample usage:

```
Bug bugsy = new Bug(10);
bugsy.move(); // now the position is 11
bugsy.turn();
bugsy.move(); // now the position is 10
```

Your `BugTester` should construct a bug, make it move and turn a few times, and print the actual and expected position.

★★ **P2.21** Implement a class `Moth` that models a moth flying across a straight line. The moth has a position, the distance from a fixed origin. When the moth moves toward a point of light, its new position is halfway between its old position and the position of the light source. Supply a constructor

```
public Moth(double initialPosition)
```

and methods

```
public void moveToLight(double lightPosition)
public double getPosition()
```

Your `MothTester` should construct a moth, move it toward a couple of light sources, and check that the moth's position is as expected.

★G **P2.22** Write a graphics program that draws your name in red, contained inside a blue rectangle. Provide a class `NameViewer` and a class `NameComponent`.

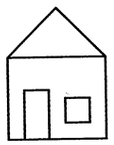
★G **P2.23** Write a program to plot the following face.

Provide a class `FaceViewer` and a class `FaceComponent`.

★G **P2.24** Draw a "bull's eye"—a set of concentric rings in alternating black and white colors. *Hint:* Fill a black circle, then fill a smaller white circle on top, and so on.

Your program should be composed of classes `BullsEye`, `BullsEyeComponent`, and `BullsEyeViewer`.

★★G **P2.25** Write a program that draws a picture of a house. It could be as simple as the accompanying figure, or if you like, make it more elaborate (3-D, skyscraper, marble columns in the entryway, whatever).



Implement a class `House` and supply a method `draw(Graphics2D g2)` that draws the house.

★★G **P2.26** Extend Exercise P2.25 by supplying a `House` constructor for specifying the position and size. Then populate your screen with a few houses of different sizes.

★★G **P2.27** Change the car viewer program in Section 2.16 to make the cars appear in different colors. Each `Car` object should store its own color. Supply modified `Car` and `CarComponent` classes.

★★G **P2.28** Change the `Car` class so that the size of a car can be specified in the constructor. Change the `CarComponent` class to make one of the cars appear twice the size of the original example.

★★G **P2.29** Write a program to plot the string "HELLO", using only lines and circles. Do not call `drawString`, and do not use `System.out`. Make classes `LetterH`, `LetterE`, `LetterL`, and `LetterO`.

★★G **P2.30** Write a program that displays the Olympic rings. Color the rings in the Olympic colors.

Provide a class `OlympicRingViewer` and a class `OlympicRingComponent`.

Programming Projects

Project 2.1 The `GregorianCalendar` class describes a point in time, as measured by the Gregorian calendar, the standard calendar that is commonly used throughout the world today. You construct a `GregorianCalendar` object from a year, month, and day of the month, like this:

```
GregorianCalendar cal = new GregorianCalendar(); // Today's date
GregorianCalendar eckertsBirthday = new GregorianCalendar(1919,
      Calendar.APRIL, 9);
```

Use the values `Calendar.JANUARY . . . Calendar.DECEMBER` to specify the month.

The `add` method can be used to add a number of days to a `GregorianCalendar` object:

```
cal.add(Calendar.DAY_OF_MONTH, 10); // Now cal is ten days from today
```

This is a mutator method—it changes the `cal` object.

The `get` method can be used to query a given `GregorianCalendar` object:

```
int dayOfMonth = cal.get(Calendar.DAY_OF_MONTH);
int month = cal.get(Calendar.MONTH);
int year = cal.get(Calendar.YEAR);
int weekday = cal.get(Calendar.DAY_OF_WEEK);
   // 1 is Sunday, 2 is Monday, . . . , 7 is Saturday
```

Your task is to write a program that prints the following information:

- The date and weekday that is 100 days from today
- The weekday of your birthday
- The date that is 10,000 days from your birthday

Use the birthday of a computer scientist if you don't want to reveal your own birthday.

Project 2.2 In this project, you will enhance the `BankAccount` class and see how abstraction and encapsulation enable evolutionary changes to software.

Begin with a simple enhancement: charging a fee for every deposit and withdrawal. Supply a mechanism for setting the fee and modify the `deposit` and `withdraw` methods so that the fee is levied. Test your resulting class and check that the fee is computed correctly.

Now make a more complex change. The bank will allow a fixed number of free transactions (deposits or withdrawals) every month, and charge for transactions exceeding the free allotment. The charge is not levied immediately but at the end of the month.

Supply a new method `deductMonthlyCharge` to the `BankAccount` class that deducts the monthly charge and resets the transaction count. (*Hint:* Use `Math.max(actual transaction count, free transaction count)` in your computation.)

Produce a test program that verifies that the fees are calculated correctly over several months.

Answers to Self-Check Questions

1. `"Mississippi".length()`
2. The implicit parameter is `"Hello"`. There is no explicit parameter. The return value is 5.
3. It is not legal. The implicit parameter `"Hello"` has type `String`. The `println` method is not a method of the `String` class.
4. Only the first two are legal identifiers.
5. `String myName = "John Q. Public";`
6. `greeting = "Hello, Nina!";`

   Note that

   `String greeting = "Hello, Nina!";`

   is not the right answer—that statement declares a new variable.
7. `new Rectangle(90, 90, 20, 20)`
8. 0
9. `toUpperCase`
10. `"Hello, Space !"`—only the leading and trailing spaces are trimmed.
11. Add the statement `import java.util.Random;` at the top of your program.
12. `x: 30, y: 25`
13. Because the `translate` method doesn't modify the shape of the rectangle.
14.
```
public void reset()
{
   value = 0;
}
```
15. You can only access them by invoking the methods of the `Clock` class.
16. `harrysChecking.withdraw(harrysChecking.getBalance())`
17. The `withdraw` method has return type `void`. It doesn't return a value. Use the `getBalance` method to obtain the balance after the withdrawal.

18. Add an `accountNumber` parameter to the constructors, and add a `getAccountNumber` method. There is no need for a `setAccountNumber` method—the account number never changes after construction.
19. An instance variable

    `private int accountNumber;`

    needs to be added to the class.
20. Because the `balance` instance variable is accessed from the `main` method of `BankRobber`. The compiler will report an error because `main` is not a method of the `BankAccount` class.
21.
```
public int getWidth()
{
   return width;
}
```
22. There is more than one correct answer. One possible implementation is as follows:
```
public void translate(int dx, int dy)
{
   int newx = x + dx;
   x = newx;
   int newy = y + dy;
   y = newy;
}
```
23. One `BankAccount` object, no `BankAccountTester` object. The purpose of the `BankAccountTester` class is merely to hold the `main` method.
24. In those environments, you can issue interactive commands to construct `BankAccount` objects, invoke methods, and display their return values.
25. Variables of both categories belong to methods—they come alive when the method is called, and they die when the method exits. They differ in their initialization. Parameter variables are initialized with the call values; local variables must be explicitly initialized.
26. After computing the change due, `payment` and `purchase` were set to zero. If the method returned `payment - purchase`, it would always return zero.
27. Now `greeting` and `greeting2` both refer to the same `String` object.
28. Both variables still refer to the same string, and the string has not been modified. Note that the `toUpperCase` method constructs a new string that contains uppercase characters, leaving the original string unchanged.
29. One implicit parameter, called `this`, of type `BankAccount`, and one explicit parameter, called `amount`, of type `double`.
30. It is not a legal expression. `this` is of type `BankAccount` and the `BankAccount` class has no instance variable named `amount`.
31. No implicit parameter—the `main` method is not invoked on any object—and one explicit parameter, called `args`.
32. Modify the `EmptyFrameViewer` program as follows:
```
frame.setSize(300, 300);
frame.setTitle("Hello, World!");
```
33. Construct two `JFrame` objects, set each of their sizes, and call `setVisible(true)` on each of them.
34. `Rectangle box = new Rectangle(5, 10, 20, 20);`

35. Replace the call to `box.translate(15, 25)` with

```
box = new Rectangle(20, 35, 20, 20);
```

36. The compiler complains that `g` doesn't have a `draw` method.

37. `g2.draw(new Ellipse2D.Double(75, 75, 50, 50));`

38.
```
Line2D.Double segment1 = new Line2D.Double(0, 0, 10, 30);
g2.draw(segment1);
Line2D.Double segment2 = new Line2D.Double(10, 30, 20, 0);
g2.draw(segment2);
```

39. `g2.drawString("V", 0, 30);`

40. `0, 0, 255`

41. First fill a big red square, then fill a small yellow square inside:

```
g2.setColor(Color.RED);
g2.fill(new Rectangle(0, 0, 200, 200));
g2.setColor(Color.YELLOW);
g2.fill(new Rectangle(50, 50, 100, 100));
```

42. `CarComponent`

43. In the `draw` method of the `Car` class, call

```
g2.fill(frontTire);
g2.fill(rearTire);
```

44. Double all measurements in the `draw` method of the `Car` class.