

# JAVA™

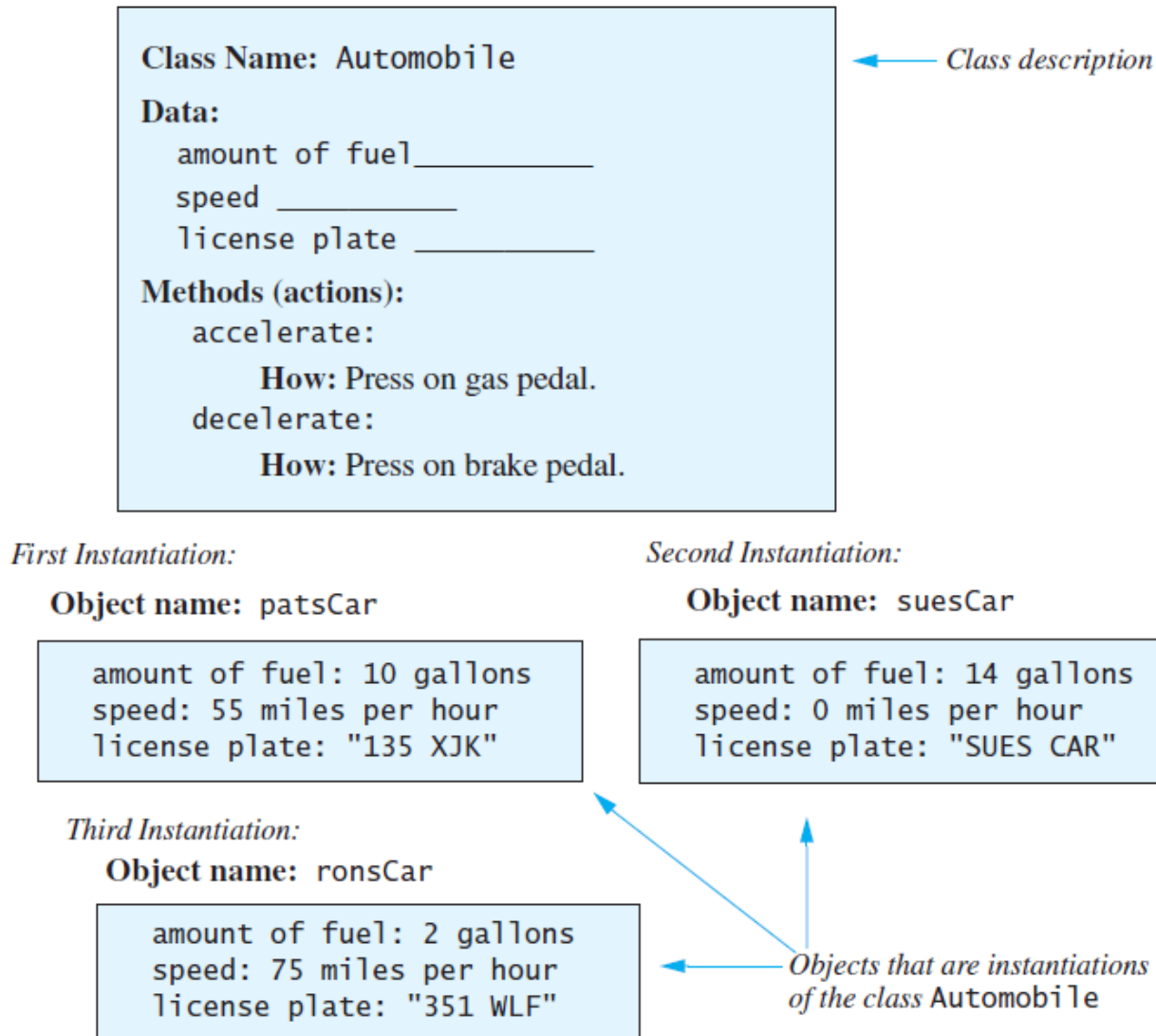
AN INTRODUCTION TO  
PROBLEM SOLVING  
AND PROGRAMMING

7TH EDITION

WALTER SAVITCH

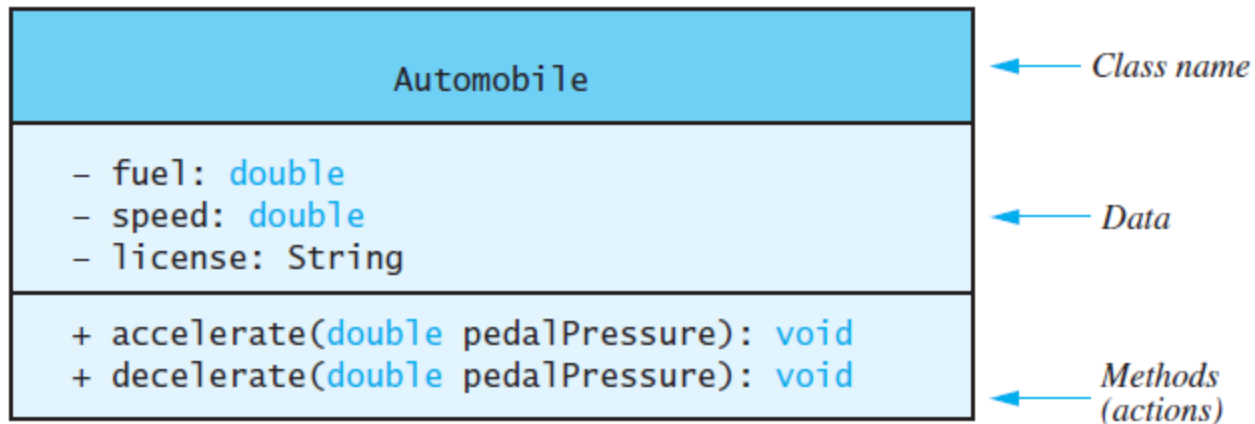
## Defining Classes and Methods 5

**FIGURE 5.1 A Class as a Blueprint**



**FIGURE 5.2** A Class Outline as a UML Class Diagram

---



## LISTING 5.1 Definition of a Dog Class

---

```
public class Dog
{
    public String name;
    public String breed;
    public int age;
    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Breed: " + breed);
        System.out.println("Age in calendar years: " +
                           age);
        System.out.println("Age in human years: " +
                           getAgeInHumanYears());
        System.out.println();
    }
    public int getAgeInHumanYears()
    {
        int humanAge = 0;
        if (age <= 2)
        {
            humanAge = age * 11;
        }
        else
        {
            humanAge = 22 + ((age-2) * 5);
        }
        return humanAge;
    }
}
```

*Later in this chapter we will see that the modifier **public** for instance variables should be replaced with **private**.*

## LISTING 5.2 Using the Dog Class and Its Methods

---

```
public class DogDemo
{
    public static void main(String[] args)
    {
        Dog balto = new Dog();
        balto.name = "Balto";
        balto.age = 8;
        balto.breed = "Siberian Husky";
        balto.writeOutput();

        Dog scooby = new Dog();
        scooby.name = "Scooby";
        scooby.age = 42;
        scooby.breed = "Great Dane";
        System.out.println(scooby.name + " is a " +
                           scooby.breed + ".");
        System.out.print("He is " + scooby.age +
                           " years old, or ");
        int humanYears = scooby.getAgeInHumanYears();
        System.out.println(humanYears + " in human years.");
    }
}
```

---

### Sample Screen Output

```
Name: Balto
Breed: Siberian Husky
Age in calendar years: 8
Age in human years: 52

Scooby is a Great Dane.
He is 42 years old, or 222 in human years.
```

### LISTING 5.3 A Species Class Definition—First Attempt (part 1 of 2)

```
import java.util.Scanner;
public class SpeciesFirstTry
{
    public String name;
    public int population;
    public double growthRate;
```

*We will give a better version of this class later in this chapter.*

*Later in this chapter you will see that the modifier **public** for instance variables should be replaced with **private**.*

```
    public void readInput()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("What is the species' name?");
        name = keyboard.nextLine();
        System.out.println("What is the population of the " +
                           "species?");
        population = keyboard.nextInt();
```

```

        System.out.println("Enter growth rate " +
                           "(% increase per year):");
        growthRate = keyboard.nextDouble();
    }
    public void writeOutput()
    {
        System.out.println("Name = " + name);
        System.out.println("Population = " + population);
        System.out.println("Growth rate = " + growthRate + "%");
    }
    public int getPopulationIn10()
    {
        int result = 0;
        double populationAmount = population;
        int count = 10;
        while ((count > 0) && (populationAmount > 0))
        {
            populationAmount = populationAmount +
                               (growthRate / 100) *
                               populationAmount;

            count--;
        }
        if (populationAmount > 0)
            result = (int)populationAmount;
        return result;
    }
}

```

## LISTING 5.4 Using the Species Class and Its Methods (part 1 of 2)

---

```
public class SpeciesFirstTryDemo
{
    public static void main(String[] args)
    {
        SpeciesFirstTry speciesOfTheMonth = new SpeciesFirstTry();
        System.out.println("Enter data on the Species of "+
                           "the Month:");

        speciesOfTheMonth.readInput();
        speciesOfTheMonth.writeOutput();
        int futurePopulation =
            speciesOfTheMonth.getPopulationIn10();
        System.out.println("In ten years the population will be "
                           + futurePopulation);
        //Change the species to show how to change
        //the values of instance variables:
        speciesOfTheMonth.name = "Klingon ox";
        speciesOfTheMonth.population = 10;
        speciesOfTheMonth.growthRate = 15;
        System.out.println("The new Species of the Month:");
        speciesOfTheMonth.writeOutput();
        System.out.println("In ten years the population will "
                           "be " + speciesOfTheMonth.getPopulationIn10());
    }
}
```



### *Sample Screen Output*

```
Enter data on the Species of the Month:
What is the species' name?
Ferengie fur ball
What is the population of the species?
1000
Enter growth rate (% increase per year):
-20.5
Name = Ferengie fur ball
Population = 1000
Growth rate = 20.5%
In ten years the population will be 100
The new Species of the Month:
Name = Klingon ox
Population = 10
Growth rate = 15.0%
In ten years the population will be 40
```

## LISTING 5.5 Local Variables

*This class definition is in a file named BankAccount.java.*

```
/**
 * This class is used in the program LocalVariablesDemoProgram.
 */
public class BankAccount
{
    public double amount;
    public double rate;
    public void showNewBalance()
    {
        double newAmount = amount + (rate / 100.0) * amount;
        System.out.println("With interest added, the new amount
                           is $" + newAmount);
    }
}
```

*This does not change the value of the variable newAmount in main.*

*Two different variables named newAmount*

*This program is in a file named LocalVariableDemoProgram.java.*

```
/**
 * A toy program to illustrate how local variables behave.
 */
public class LocalVariablesDemoProgram
{
    public static void main(String[] args)
    {
        BankAccount myAccount = new BankAccount();
        myAccount.amount = 100.00;
        myAccount.rate = 5;

        double newAmount = 800.00;
        myAccount.showNewBalance();
        System.out.println("I wish my new amount were $" +
                           newAmount);
    }
}
```

### **Screen Output**

*Chapter 6 will fix the appearance of dollar amounts.*

```
With interest added, the new amount is $105.0  
I wish my new amount were $800.0
```

---

## LISTING 5.6 A Method That Has a Parameter

---

```
import java.util.Scanner;  
public class SpeciesSecondTry  
{
```

*We will give an even better version of the class later in the chapter.*

<The declarations of the instance variables name, population, and growthRate are the same as in Listing 5.3.>

<The definitions of the methods readInput and writeOutput are the same as in Listing 5.3.>

```
/**
```

*Returns the projected population of the receiving object after the specified number of years.*

```
*/
```

```
public int predictPopulation(int years)  
{
```

```
    int result = 0;
```

```
    double populationAmount = population;
```

```
    int count = years;
```

```
    while ((count > 0) && (populationAmount > 0))  
    {
```

```
        populationAmount = (populationAmount +  
            (growthRate / 100) * populationAmount);  
        count--;
```

```
    }
```

```
    if (populationAmount > 0)
```

```
        result = (int)populationAmount;
```

```
    return result;
```

```
}
```

```
}
```

## LISTING 5.7 Using a Method That Has a Parameter

---

```
/**
 Demonstrates the use of a parameter
 with the method predictPopulation.
 */
public class SpeciesSecondTryDemo
{
    public static void main(String[] args)
    {
        SpeciesSecondTry speciesOfTheMonth = new
            SpeciesSecondTry();
        System.out.println("Enter data on the Species of the " +
            "Month:");
        speciesOfTheMonth.readInput();
        speciesOfTheMonth.writeOutput();
        int futurePopulation =
            speciesOfTheMonth.predictPopulation(10);
        System.out.println("In ten years the population will be " +
            futurePopulation);
        //Change the species to show how to change
        //the values of instance variables:
        speciesOfTheMonth.name = "Klingon ox";
        speciesOfTheMonth.population = 10;
        speciesOfTheMonth.growthRate = 15;
        System.out.println("The new Species of the Month:");
        speciesOfTheMonth.writeOutput();
        System.out.println("In ten years the population will be " +
            speciesOfTheMonth.predictPopulation(10));
    }
}
```

---

### Sample Screen Output

*The output is exactly the same  
as in Listing 5.4.*

## LISTING 5.8 A Class with Private Instance Variables

---

```
import java.util.Scanner;  
public class SpeciesThirdTry  
{
```

```
    private String name;  
    private int population;  
    private double growthRate;
```

<The definitions of the methods readInput, writeOutput, and predictPopulation are the same as in Listing 5.3 and Listing 5.6.>

```
}
```

---

*We will give an even better version of this class later in the chapter.*

## LISTING 5.9 A Class of Rectangles

---

```
/**
Class that represents a rectangle.
*/
public class Rectangle
{
    private int width;
    private int height;
    private int area;

    public void setDimensions(int newWidth, int newHeight)
    {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea()
    {
        return area;
    }
}
```

---

## LISTING 5.10 Another Class of Rectangles

---

```
/**  
 Another class that represents a rectangle.  
 */  
public class Rectangle2  
{  
    private int width;  
    private int height;  
    public void setDimensions(int newWidth, int newHeight)  
    {  
        width = newWidth;  
        height = newHeight;  
    }  
    public int getArea()  
    {  
        return width * height;  
    }  
}
```

---



## LISTING 5.11 A Class with Accessor and Mutator Methods

```
import java.util.Scanner;
public class SpeciesFourthTry
{
```

*Yes, we will define an even better version of this class later.*

```
    private String name;
    private int population;
    private double growthRate;
```

<The definitions of the methods readInput, writeOutput, and predictPopulation go here. They are the same as in Listing 5.3 and Listing 5.6.>

```
    public void setSpecies(String newName, int newPopulation,
                           double newGrowthRate)
```

```
    {
        name = newName;
        if (newPopulation >= 0)
            population = newPopulation;
        else
        {
            System.out.println(
                "ERROR: using a negative population.");
            System.exit(0);
        }
        growthRate = newGrowthRate;
    }
```

```
    public String getName()
```

```
    {
        return name;
    }
```

```
    public int getPopulation()
```

```
    {
        return population;
    }
```

```
    public double getGrowthRate()
```

```
    {
        return growthRate;
    }
```

```
}
```

*A mutator method can check to make sure that instance variables are set to proper values.*

## LISTING 5.12 Using a Mutator Method (part 1 of 2)

---

```
import java.util.Scanner;
/**
 Demonstrates the use of the mutator method setSpecies.
 */
public class SpeciesFourthTryDemo
{
    public static void main(String[] args)
    {
        SpeciesFourthTry speciesOfTheMonth =
            new SpeciesFourthTry();
        System.out.println("Enter number of years to project:");
        Scanner keyboard = new Scanner(System.in);
        int numberOfYears = keyboard.nextInt();

        System.out.println(
            "Enter data on the Species of the Month:");
        speciesOfTheMonth.readInput();
        speciesOfTheMonth.writeOutput();

        int futurePopulation =
            speciesOfTheMonth.predictPopulation(numberOfYears);
        System.out.println("In " + numberOfYears +
            " years the population will be " +
            futurePopulation);
        //Change the species to show how to change
        //the values of instance variables:
        speciesOfTheMonth.setSpecies("Klingon ox", 10, 15);
        System.out.println("The new Species of the Month:");
        speciesOfTheMonth.writeOutput();

        futurePopulation =
            speciesOfTheMonth.predictPopulation(numberOfYears);
        System.out.println("In " + numberOfYears +
            " years the population will be " +
            futurePopulation);
    }
}
```

### *Sample Screen Output*

Enter number of years to project:

10

Enter data on the Species of the Month:

What is the species' name?

Ferengie fur ball

### LISTING 5.13 The Purchase Class (part 1 of 3)

---

```
import java.util.Scanner;

/**
Class for the purchase of one kind of item, such as 3 oranges.
Prices are set supermarket style, such as 5 for $1.25.
*/
public class Purchase
{
    private String name;
    private int groupCount;    //Part of a price, like the 2 in
                                //2 for $1.99.
    private double groupPrice; //Part of a price, like the $1.99
                                // in 2 for $1.99.
    private int numberBought;  //Number of items bought.
    public void setName(String newName)
    {
        name = newName;
    }

    /**
Sets price to count pieces for $costForCount.
For example, 2 for $1.99.
*/
    public void setPrice(int count, double costForCount)
    {
        if ((count <= 0) || (costForCount <= 0))
        {
            System.out.println("Error: Bad parameter in " +
                                "setPrice.");
            System.exit(0);
        }
        else
        {
            groupCount = count;
            groupPrice = costForCount;
        }
    }
}
```

```
public void setNumberBought(int number)
{
    if (number <= 0)
    {
        System.out.println("Error: Bad parameter in " +
                           "setNumberBought.");
        System.exit(0);
    }
    else
        numberBought = number;
}
```

```

/**
Reads from keyboard the price and number of a purchase.
*/
public void readInput()
{
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter name of item you are purchasing:");
    name = keyboard.nextLine();
    System.out.println("Enter price of item as two numbers.");
    System.out.println("For example, 3 for $2.99 is entered as");
    System.out.println("3 2.99");
    System.out.println("Enter price of item as two numbers, " +
        "now:");
    groupCount = keyboard.nextInt();
    groupPrice = keyboard.nextDouble();

    while ((groupCount <= 0) || (groupPrice <= 0))
    { //Try again:
        System.out.println("Both numbers must " +
            "be positive. Try again.");
        System.out.println("Enter price of " +
            "item as two numbers.");
        System.out.println("For example, 3 for " +
            "$2.99 is entered as");
        System.out.println("3 2.99");
        System.out.println(
            "Enter price of item as two numbers, now:");
        groupCount = keyboard.nextInt();
        groupPrice = keyboard.nextDouble();
    }
    System.out.println("Enter number of items purchased:");
    numberBought = keyboard.nextInt();

    while (numberBought <= 0)
    { //Try again:
        System.out.println("Number must be positive. " +
            "Try again.");
        System.out.println("Enter number of items purchased:");
        numberBought = keyboard.nextInt();
    }
}

```

```

/**
Displays price and number being purchased.
 */
public void writeOutput()
{
    System.out.println(numberBought + " " + name);
    System.out.println("at " + groupCount +
        " for $" + groupPrice);
}
public String getName()
{
    return name;
}
public double getTotalCost()
{
    return (groupPrice / groupCount) * numberBought;
}
public double getUnitCost()
{
    return groupPrice / groupCount;
}
public int getNumberBought()
{
    return numberBought;
}
}

```

## LISTING 5.14 Use of the Purchase Class

---

```
public class PurchaseDemo
{
    public static void main(String[] args)
    {
        Purchase oneSale = new Purchase();
        oneSale.readInput();
        oneSale.writeOutput();
        System.out.println("Cost each $" + oneSale.getUnitCost());
        System.out.println("Total cost $" +
                           oneSale.getTotalCost());
    }
}
```

### Sample Screen Output

```
Enter name of item you are purchasing:
pink grapefruit
Enter price of item as two numbers.
For example, 3 for $2.99 is entered as
3 2.99
Enter price of item as two numbers, now:
4 5.00
Enter number of items purchased:
0
Number must be positive. Try again.
Enter number of items purchased:
3
3 pink grapefruit
at 4 for $5.0
Cost each $1.25
Total cost $3.75
```



## LISTING 5.15 Methods Calling Other Methods

---

```
import java.util.Scanner;
public class Oracle
{
    private String oldAnswer = "The answer is in your heart.";
    private String newAnswer;
    private String question;

    public void chat()
    {
        System.out.print("I am the oracle. ");
        System.out.println("I will answer any one-line question.");
        Scanner keyboard = new Scanner(System.in);
        String response;
        do
        {
            answer();
            System.out.println("Do you wish to ask " +
                               "another question?");
            response = keyboard.next();
        } while (response.equalsIgnoreCase("yes"));
        System.out.println("The oracle will now rest.");
    }
}
```

```

private void answer()
{
    System.out.println("What is your question?");
    Scanner keyboard = new Scanner(System.in);
    question = keyboard.nextLine();
    seekAdvice();
    System.out.println("You asked the question:");
    System.out.println(" " + question);
    System.out.println("Now, here is my answer:");
    System.out.println(" " + oldAnswer);
    update();
}
private void seekAdvice()
{
    System.out.println("Hmm, I need some help on that.");
    System.out.println("Please give me one line of advice.");
    Scanner keyboard = new Scanner(System.in);
    newAnswer = keyboard.nextLine();
    System.out.println("Thank you. That helped a lot.");
}
private void update()
{
    oldAnswer = newAnswer;
}
}

```

## LISTING 5.16 Oracle Demonstration Program (part 1 of 2)

---

```
public class OracleDemo
{
    public static void main(String[] args)
    {
        Oracle delphi = new Oracle();
        delphi.chat();
    }
}
```

---

### *Sample Screen Output*

```
I am the oracle. I will answer any one-line question.
What is your question?
What time is it?
Hmm, I need some help on that.
Please give me one line of advice.
Seek and ye shall find the answer.
Thank you. That helped a lot.
You asked the question:
    What time is it?
Now, here is my answer:
    The answer is in your heart.
Do you wish to ask another question?
yes
What is your question?
What is the meaning of life?
Hmm, I need some help on that.
```

Please give me one line of advice.

Ask the car guys.

Thank you. That helped a lot.

You asked the question:

What is the meaning of life?

Now, here is my answer:

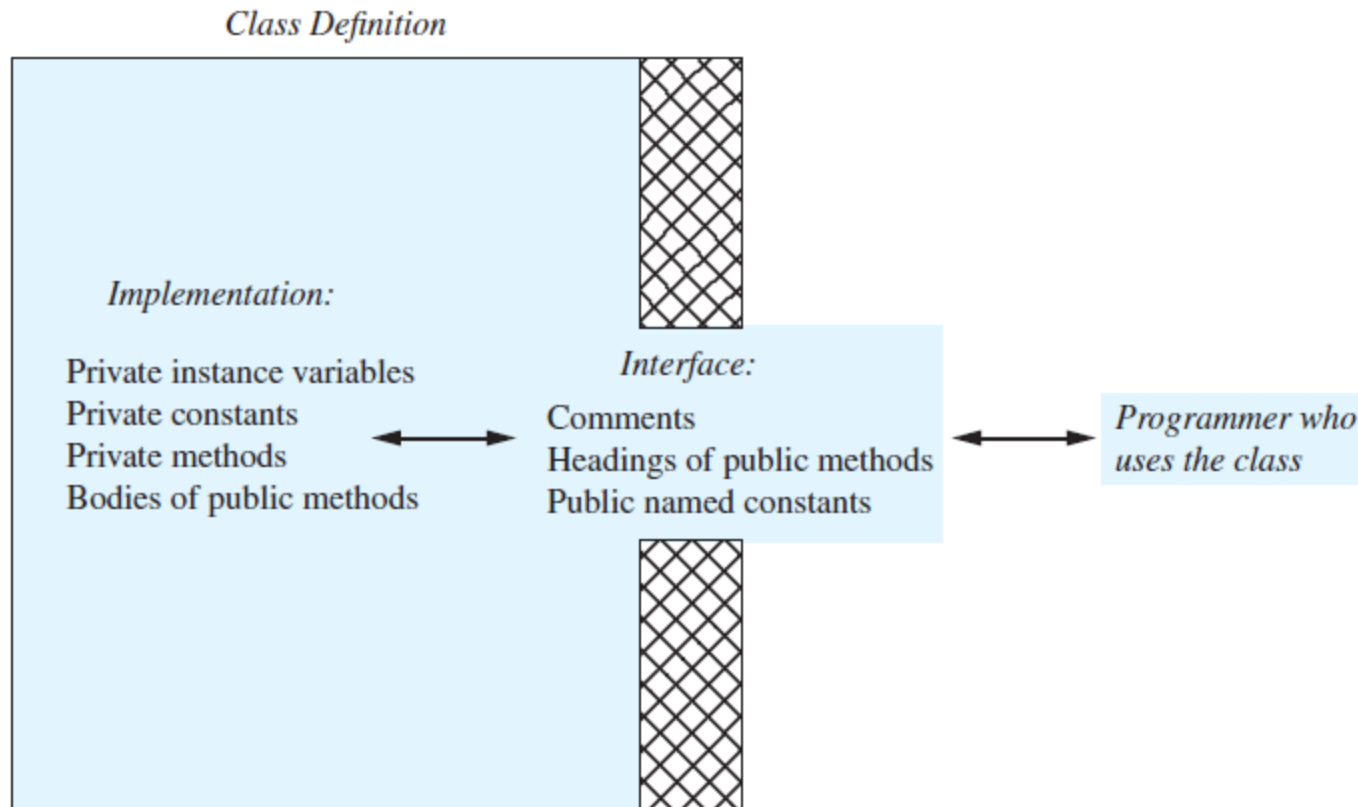
Seek and ye shall find the answer.

Do you wish to ask another question?

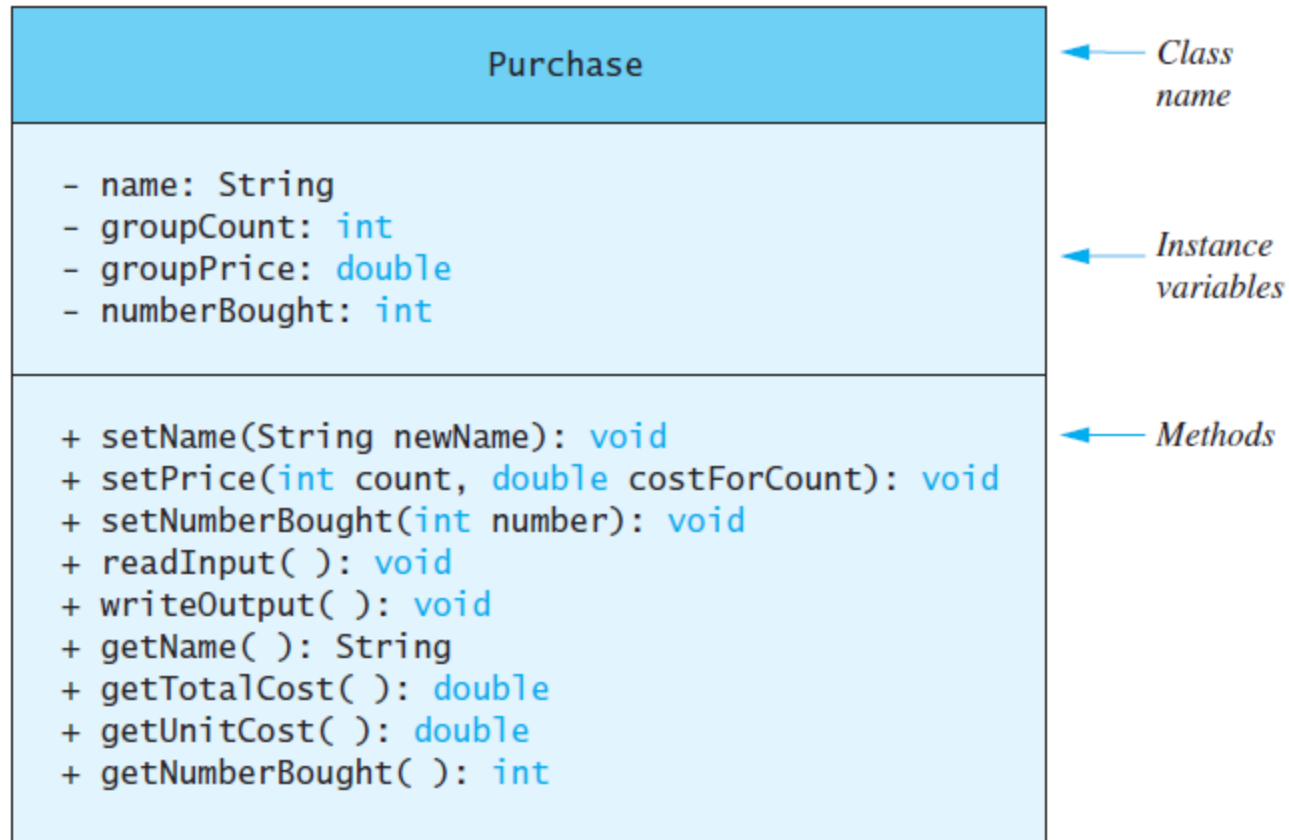
no

The oracle will now rest.

**FIGURE 5.3** A Well-Encapsulated Class Definition



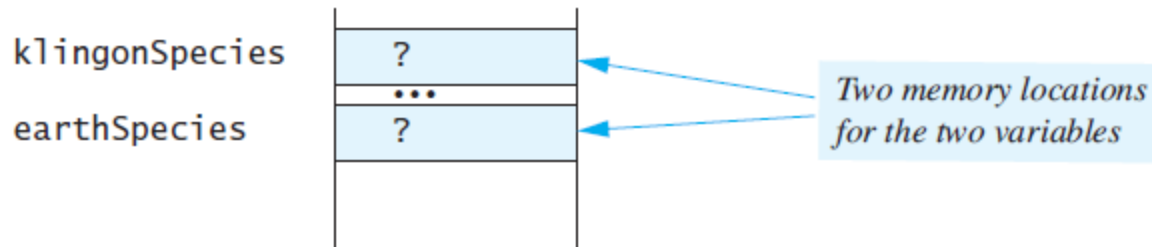
**FIGURE 5.4** A UML Class Diagram for the Class Purchase (Listing 5.13)



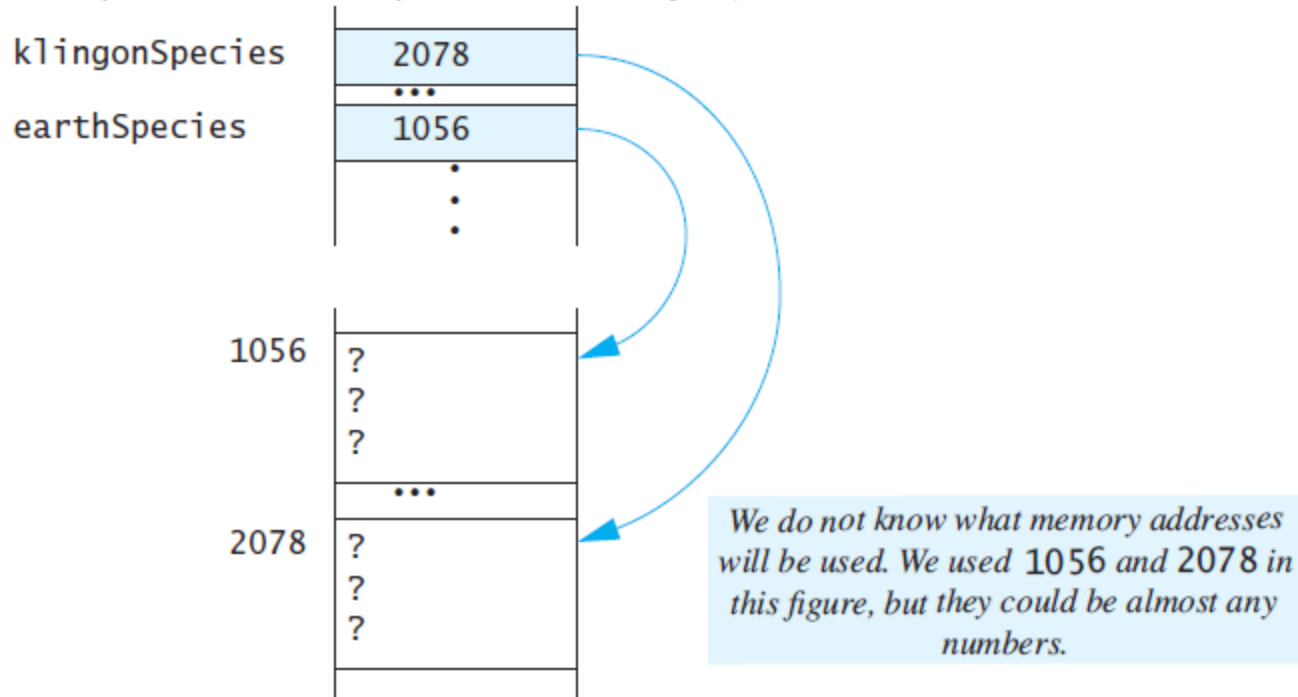
*A minus sign (-) means the member is private.  
A plus sign (+) means the member is public.*

**FIGURE 5.5 Behavior of Class Variables**

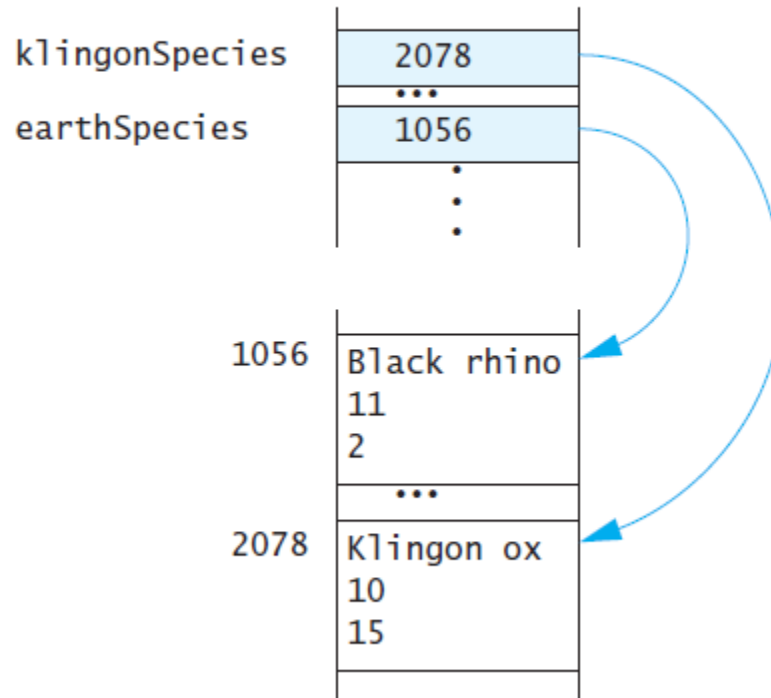
```
SpeciesFourthTry klingonSpecies, earthSpecies;
```



```
klingonSpecies = new SpeciesFourthTry();  
earthSpecies = new SpeciesFourthTry();
```



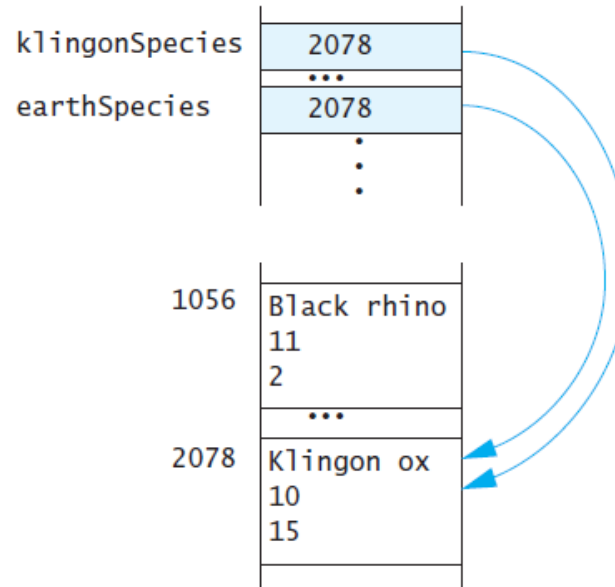
```
klingsonSpecies.setSpecies("Klingson ox", 10, 15);  
earthSpecies.setSpecies("Black rhino", 11, 2);
```





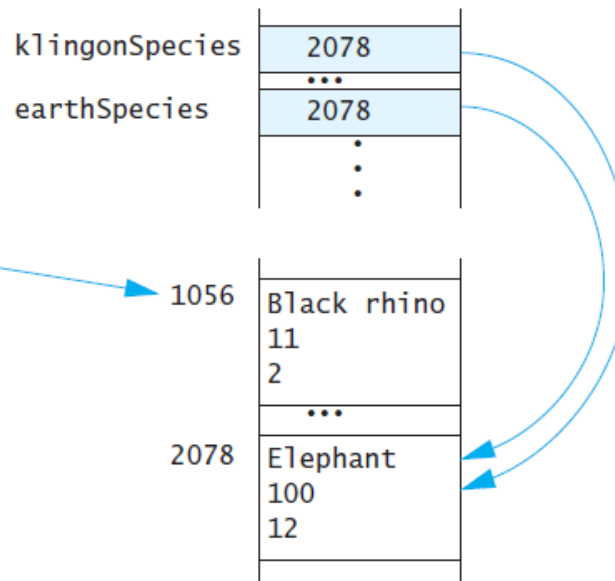
```
earthSpecies = klingonSpecies;
```

*klingonSpecies and earthSpecies are now two names for the same object.*



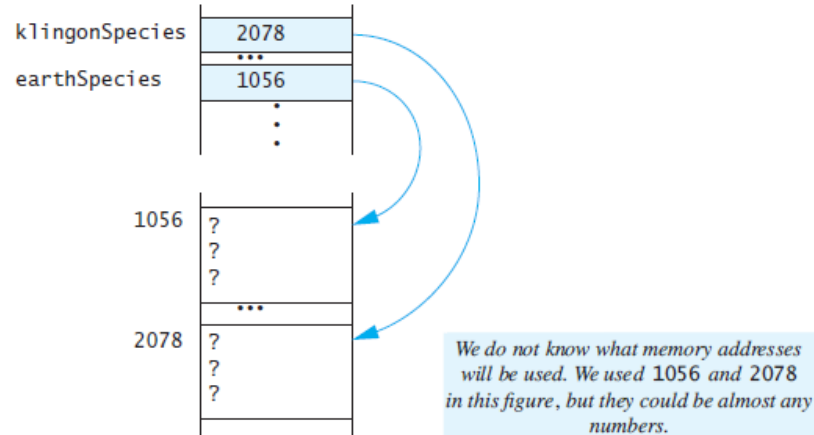
```
earthSpecies.setSpecies("Elephant", 100, 12);
```

*This is just garbage that is not accessible to the program.*

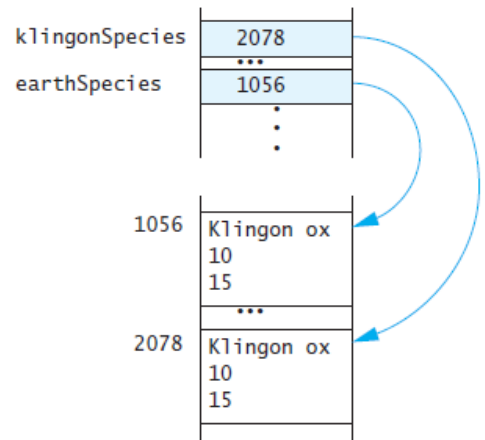


**FIGURE 5.6 The Dangers of Using == with Objects**

```
klingsonSpecies = new SpeciesFourthTry();  
earthSpecies = new SpeciesFourthTry();
```



```
klingsonSpecies.setSpecies("Klingon ox", 10, 15);  
earthSpecies.setSpecies("Klingon ox", 10, 15);
```



```
if (klingsonSpecies == earthSpecies)  
    System.out.println("They are EQUAL.");  
else  
    System.out.println("They are NOT equal.");
```

*The output is They are Not equal, because 2078 is not equal to 1056.*

## LISTING 5.17 Defining an equals Method

---

```
import java.util.Scanner;
public class Species
{
    private String name;
    private int population;
    private double growthRate;
```

<The definition of the methods readInput, writeOutput, and predictPopulation go here. They are the same as in Listing 5.3 and Listing 5.6.>

<The definition of the methods setSpecies, getName, getPopulation, and getGrowthRate go here. They are the same as in Listing 5.11.>

```
public boolean equals(Species otherObject)
{
    return (this.name.equalsIgnoreCase(otherObject.name)) &&
           (this.population == otherObject.population) &&
           (this.growthRate == otherObject.growthRate);
}
}
```

`equalsIgnoreCase` is a method of the class `String`.

---

### **LISTING 5.18** Demonstrating an equals Method *(part 1 of 2)*

---

```
public class SpeciesEqualsDemo
{
    public static void main(String[] args)
    {
        Species s1 = new Species(), s2 = new Species();
        s1.setSpecies("Klingon ox", 10, 15);
        s2.setSpecies("Klingon ox", 10, 15);

        if (s1 == s2)
            System.out.println("Match with ==.");
        else
            System.out.println("Do Not match with ==.");
        if (s1.equals(s2))
            System.out.println("Match with the method " +
                               "equals.");
        else
            System.out.println("Do Not match with the method " +
                               "equals.");
        System.out.println("Now change one Klingon ox to " +
                           "lowercase.");
    }
}
```

```
s2.setSpecies("klington ox", 10, 15); //Use lowercase  
if (s1.equals(s2))  
    System.out.println("Match with the method equals.");  
else  
    System.out.println("Do Not match with the method " +  
                        "equals.");  
}  
}
```

---

### Screen Output

```
Do Not match with ==.  
Match with the method equals.  
Now change one Klinton ox to lowercase.  
Match with the method equals.
```

---

## LISTING 5.19 The Complete Species Class (part 1 of 2)

---

```
import java.util.Scanner;
/**
Class for data on endangered species.
*/
public class Species
{
    private String name;
    private int population;
    private double growthRate;

    public void readInput()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("What is the species' name?");
        name = keyboard.nextLine();

        System.out.println(
            "What is the population of the species?");
        population = keyboard.nextInt();
        while (population < 0)
        {
            System.out.println("Population cannot be negative.");
            System.out.println("Reenter population:");
            population = keyboard.nextInt();
        }
        System.out.println(
            "Enter growth rate (% increase per year):");
        growthRate = keyboard.nextDouble();
    }
}
```

*This is the same class definition as in Listing 5.17, but with all the details shown.*

```

public void writeOutput()
{
    System.out.println("Name = " + name);
    System.out.println("Population = " + population);
    System.out.println("Growth rate = " + growthRate + "%");
}
/**
Precondition: years is a nonnegative number.
Returns the projected population of the receiving object
after the specified number of years.
 */
public int predictPopulation(int years)
{
    int result = 0;
    double populationAmount = population;

    int count = years;
    while ((count > 0) && (populationAmount > 0))
    {
        populationAmount = (populationAmount +
                             (growthRate / 100) *
                             populationAmount);

        count--;
    }
    if (populationAmount > 0)
        result = (int)populationAmount;
    return result;
}

```

```

public void setSpecies(String newName, int newPopulation,
                        double newGrowthRate)
{
    name = newName;
    if (newPopulation >= 0)
        population = newPopulation;
    else
    {
        System.out.println("ERROR: using a negative " +
                           "population.");
        System.exit(0);
    }
    growthRate = newGrowthRate;
}
public String getName()
{
    return name;
}
public int getPopulation()
{
    return population;
}
public double getGrowthRate()
{
    return growthRate;
}
public boolean equals(Species otherObject)
{
    return (name.equalsIgnoreCase(otherObject.name)) &&
           (population == otherObject.population) &&
           (growthRate == otherObject.growthRate);
}
}

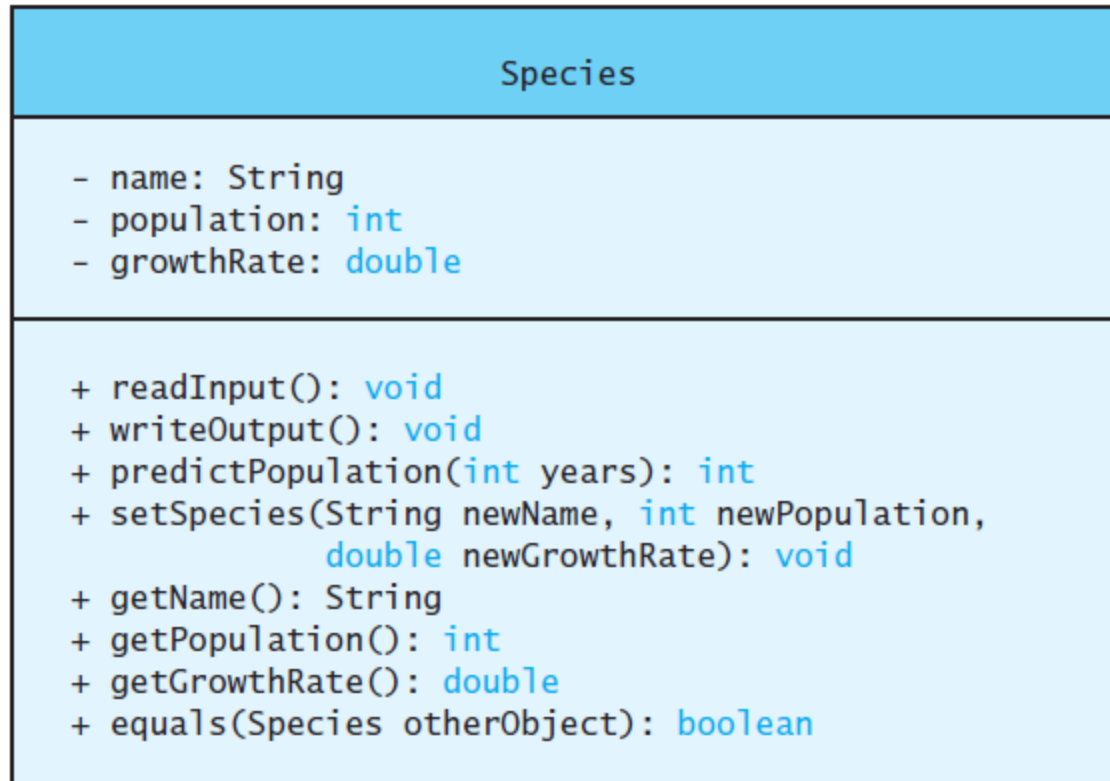
```

*This version of equals is equivalent to the version in Listing 5.17. Here, the keyword this is understood to be there implicitly.*



**FIGURE 5.7** Class Diagram for the Class Species in Listing 5.19

---



## LISTING 5.20 Sample Tests for the Species Class

---

```
public class SpeciesTest
{
    public static void main(String[] args)
    {
        Species testSpecies = new Species();

        // Test the setSpecies method
        testSpecies.setSpecies("Tribbles", 100, 50);
        if (testSpecies.getName().equals("Tribbles") &&
            (testSpecies.getPopulation() == 100) &&
            (testSpecies.getGrowthRate() >= 49.99) &&
            (testSpecies.getGrowthRate() <= 50.01))
        {
            System.out.println("Pass: setSpecies test.");
        }
        else
        {
            System.out.println("FAIL: setSpecies test.");
        }

        // Test the predictPopulation method
        if ((testSpecies.predictPopulation(-1) == 100) &&
            (testSpecies.predictPopulation(1) == 150) &&
            (testSpecies.predictPopulation(5) == 759))
        {
            System.out.println("Pass: predictPopulation test.");
        }
        else
        {
            System.out.println("FAIL: predictPopulation test.");
        }
    }
}
```

---

### *Sample Screen Output*

Pass: setSpecies test.

Pass: predictPopulation test.

---

## LISTING 5.21 A Demonstration Class (part 1 of 2)

---

```
import java.util.Scanner;
/**
This version of the class Species is only a toy example designed
to demonstrate the difference between parameters of a class type
and parameters of a primitive type.
*/
public class DemoSpecies
{
    private String name;
    private int population;
    private double growthRate;
    /**
Tries to set intValue equal to the population of this
object. But arguments of a primitive type cannot be
changed.
    */
    public void tryToChange(int intValue)
    {
        intValue = this.population;
    }
    /**
Tries to make otherObject reference this object.
But arguments of a class type cannot be replaced.
    */
    public void tryToReplace(DemoSpecies otherObject)
    {
        otherObject = this;
    }
    /**
Changes the data in otherObject to the data in this object,
which is unchanged.
    */
}
```

```
public void change(DemoSpecies otherObject)
{
    otherObject.name = this.name;
    otherObject.population = this.population;
    otherObject.growthRate = this.growthRate;
}
<The rest of the class definition is the same as that of the class
Species in Listing 5.19.>
}
```

## LISTING 5.22 Parameters of a Class Type Versus Parameters of a Primitive Type

---

```
public class ParametersDemo
{
    public static void main(String[] args)
    {
        DemoSpecies s1 = new DemoSpecies(),
                    s2 = new DemoSpecies();
        s1.setSpecies("Klingon ox", 10, 15);
        int aPopulation = 42;
        System.out.println("aPopulation BEFORE calling " +
                           "tryToChange: " + aPopulation);
        s1.tryToChange(aPopulation);
        System.out.println("aPopulation AFTER calling " +
                           "tryToChange: aPopulation);
        s2.setSpecies("Ferengie Fur Ball", 90, 56);
        System.out.println("s2 BEFORE calling tryToReplace: ");
        s2.writeOutput();
        s1.tryToReplace(s2);
        System.out.println("s2 AFTER calling tryToReplace: ");
        s2.writeOutput();
        s1.change(s2);
        System.out.println("s2 AFTER calling change: ");
        s2.writeOutput();
    }
}
```

## Screen Output

aPopulation BEFORE calling tryToChange: 42

*An argument of a primitive  
type cannot change in value.*

aPopulation AFTER calling tryToChange: 42

s2 BEFORE calling tryToReplace:

Name = Ferengie Fur Ball

Population = 90

Growth Rate = 56.0%

s2 AFTER calling tryToReplace:

*An argument of a class  
type cannot be replaced.*

Name = Ferengie Fur Ball

Population = 90

Growth Rate = 56.0%

s2 AFTER calling change:

Name = Klingon ox

*An argument of a class  
type can change in state.*

Population = 10

Growth Rate = 15.0%

**FIGURE 5.8** Some Methods in the Class `Graphics`

---

<i>Graphics_Object.drawOval(X, Y, Width, Height)</i> Draws the outline of an oval having the specified width and height at the point (X, Y).
<i>Graphics_Object.fillOval(X, Y, Width, Height)</i> Same as <code>drawOval</code> , but the oval is filled in.
<i>Graphics_Object.drawArc(X, Y, Width, Height, Start_Angle, ArcAngle)</i> Draws an arc—that is, draws part of an oval. See the graphics supplement section of Chapter 1 for details.
<i>Graphics_Object.fillArc(X, Y, Width, Height, Start_Angle, ArcAngle)</i> Same as <code>drawArc</code> , but the visible portion of the oval is filled in.
<i>Graphics_Object.drawRect(X, Y, Width, Height)</i> Draws the outline of a rectangle of the specified width and height at the point (X, Y).
<i>Graphics_Object.fillRect(X, Y, Width, Height)</i> Same as <code>drawRect</code> , but the rectangle is filled in.
<i>Graphics_Object.drawLine(X1, Y1, X2, Y2)</i> Draws a line between points (X1, Y1) and (X2, Y2).
<i>Graphics_Object.drawString(A_String, X, Y)</i> Writes the specified string starting at the point (X, Y).
<i>Graphics_Object.setColor(Color_Object)</i> Sets the color for subsequent drawings and text. The color stays in effect until it is changed by another invocation of <code>setColor</code> .



## LISTING 5.23 Using a Method for a Recurrent Subtask (part 1 of 3)

---

```
import javax.swing.JApplet;
import java.awt.Graphics;
import java.awt.Color;

public class MultipleFaces extends JApplet
{
    public static final int FACE_DIAMETER = 50;
    public static final int X_FACE0 = 10;
    public static final int Y_FACE0 = 5;

    public static final int EYE_WIDTH = 5;
    public static final int EYE_HEIGHT = 10;
    public static final int X_RIGHT_EYE0 = 20;
    public static final int Y_RIGHT_EYE0 = 15;
    public static final int X_LEFT_EYE0 = 45;
    public static final int Y_LEFT_EYE0 = Y_RIGHT_EYE0;

    public static final int NOSE_DIAMETER = 5;
    public static final int X_NOSE0 = 32;
    public static final int Y_NOSE0 = 25;

    public static final int MOUTH_WIDTH = 30;
    public static final int MOUTH_HEIGHT0 = 0;
```

```

public static final int X_MOUTH0 = 20;
public static final int Y_MOUTH0 = 35;
public static final int MOUTH_START_ANGLE = 180;
public static final int MOUTH_EXTENT_ANGLE = 180;
/**
 g is the drawing area. pos indicates the position of the
 face. As pos increases, the face is drawn lower and further
 to the right.
 */
private void drawFaceSansMouth(Graphics g, int pos)
{
    g.setColor(Color.BLACK);
    g.drawOval(X_FACE0 + 50 * pos, Y_FACE0 + 30 * pos,
               FACE_DIAMETER, FACE_DIAMETER);
    //Draw eyes:
    g.setColor(Color.BLUE);
    g.fillOval(X_RIGHT_EYE0 + 50 * pos, Y_RIGHT_EYE0 + 30 * pos,
               EYE_WIDTH, EYE_HEIGHT);
    g.fillOval(X_LEFT_EYE0 + 50 * pos, Y_LEFT_EYE0 + 30 * pos,
               EYE_WIDTH, EYE_HEIGHT);
    //Draw nose:
    g.setColor(Color.BLACK);
    g.fillOval(X_NOSE0 + 50 * pos, Y_NOSE0 + 30 * pos,
               NOSE_DIAMETER, NOSE_DIAMETER);
}

```

```

public void paint(Graphics canvas)
{
    super.paint(canvas)
    int i;
    for (i = 0; i < 5; i++)
    {//Draw one face:
        if (i % 2 == 0)//If i is even,
            { //make face yellow
                canvas.setColor(Color.YELLOW);
                canvas.fillOval(X_FACE0 + 50 * i,
                               Y_FACE0 + 30 * i,
                               FACE_DIAMETER, FACE_DIAMETER);
            }
        drawFaceSansMouth(canvas, i);
        //Draw mouth:
        canvas.setColor(Color.RED);
        canvas.drawArc(X_MOUTH0 + 50 * i, Y_MOUTH0 + 30 * i,
                      MOUTH_WIDTH, MOUTH_HEIGHT0 + 3 * i,
                      MOUTH_START_ANGLE, MOUTH_EXTENT_ANGLE);
    }
    //i == 5
}

```

```

        //Draw kissing face:
        drawFaceSansMouth(canvas, i);
        //Draw mouth in shape of a kiss:
        canvas.setColor(Color.RED);
        canvas.fillOval(X_MOUTH0 + 50 * i + 10, Y_MOUTH0 + 30 * i,
                        MOUTH_WIDTH - 20, MOUTH_WIDTH - 20);
        //Add text:
        canvas.setColor(Color.BLACK);
        canvas.drawString("Kiss, Kiss.",
                        X_FACE0 + 50 * i + FACE_DIAMETER, Y_FACE0 + 30 * i);
        //Draw blushing face:
        i++;
        //Draw face circle:
        canvas.setColor(Color.PINK);
        canvas.fillOval(X_FACE0 + 50 * i, Y_FACE0 + 30 * i,
                        FACE_DIAMETER, FACE_DIAMETER);
        drawFaceSansMouth(canvas, i);
        //Draw mouth:
        canvas.setColor(Color.RED);
        canvas.drawArc(X_MOUTH0 + 50 * i, Y_MOUTH0 + 30 * i,
                        MOUTH_WIDTH, MOUTH_HEIGHT0 + 3 * (i - 2),
                        MOUTH_START_ANGLE, MOUTH_EXTENT_ANGLE);
        //Add text:
        canvas.setColor(Color.BLACK);
        canvas.drawString("Tee Hee.",
                        X_FACE0 + 50 * i + FACE_DIAMETER, Y_FACE0 + 30 * i);
    }
}

```

Applet Output

*The drawing produced is identical to the one shown in Listing 4.9 except for some of the colors used to draw the faces.*

## Listing 5.24

## Image filter demonstration using the Graphics2D Class

```
import javax.swing.JFrame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.awt.image.RescaleOp;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;

public class Graphics2DExample extends JFrame
{
    public void paint(Graphics canvas)
    {
        super.paint(canvas);
        try
        {
            // Load image from default location on disk
            BufferedImage img = ImageIO.read(new File("java.jpg"));
            // Draw the image at coordinate 50,50
            canvas.drawImage(img, 50, 50, null);

            // Copy the image to another buffer with a
            // color model (ARGB) to support alpha blending
            // that allows translucency
            int w = img.getWidth(null);
            int h = img.getHeight(null);
            BufferedImage img2 = new
                BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);
            Graphics g = img2.getGraphics();
            g.drawImage(img, 0, 0, null);
```

```

// Create a rescale filter operation that
// makes the image 30% opaque
float[] scales = { 1f, 1f, 1f, 0.3f };
float[] offsets = new float[4];
RescaleOp rop = new RescaleOp(scales, offsets, null);
// Draw the image, applying the filter
Graphics2D g2 = (Graphics2D) canvas;
g2.drawImage(img2, rop, 150, 50);
}
catch (IOException e)
{
    System.out.println("Error reading the image.");
}
}
public Graphics2DExample()
{
    setSize(275,175);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public static void main(String[] args)
{
    Graphics2DExample guiWindow = new Graphics2DExample();
    guiWindow.setVisible(true);
}
}

```

## Application Output



## LISTING 5.25 Adding Labels to an Applet (part 1 of 2)

---

```
import javax.swing.JApplet;
import javax.swing.JLabel;
import java.awt.Color;
import java.awt.Container;
import java.awt.FlowLayout;

/**
 An applet that uses a label to display text.
 */
public class LabelDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.WHITE);
        //Create labels:
        JLabel label1 = new JLabel("Hello ");
        JLabel label2 = new JLabel("out there!");
        //Add labels:
        contentPane.setLayout(new FlowLayout());
        contentPane.add(label1);
        contentPane.add(label2);
    }
}
```



## Applet Output

