Primitive Type Values as Objects

Recall that you can use the `class Integer` to wrap `int` values in objects, the `class Double` to wrap `double` values in objects, and so on. Even though you can use the `class Integer` to wrap `int` values in objects, the `class Integer` does not provide a method to change the value of an existing `Integer` object. The same is true of other wrapper classes. That is, when passed as parameters, objects of wrapper classes have the same limitations as objects of the `class String`. If we want to pass a `String` object as a parameter and also change that object, we can use the `class StringBuffer`. However, Java does not provide any class that wraps primitive type values in objects and when passed as parameters change their values. If a method returns only one value of a primitive type, then you can write a value-returning method. However, if you encounter a situation that requires you to write a method that needs to pass more than one value of a primitive type, then you should design your own classes. Appendix D provides the definitions of such classes and shows how to use them in a program. In the next section, we introduce the user-defined `class IntClass`.

class IntClass

Recall that you can also pass `String` objects and `Integer` objects as parameters. Java provides the classes `Integer`, `Double`, `Character`, `Long`, and `Float` so that the values of the primitive data types can be treated as objects. As remarked in the preceding section, these classes wrap primitive type values. However, these classes have limitations. You can create objects of the type, say `Integer`, to store `int` values, but (as previously noted) you cannot change the values stored in the objects. Parameter passing, as you will see, is a fundamental concept in any programming language. Therefore, we have created the classes `IntClass` and `DoubleClass`, which are similar to the classes `Integer` and `Double`, respectively. You can create objects of type `IntClass` and/or `DoubleClass`, store, and/or change the values of the objects. The following table describes the data member and methods of the `class IntClass` (see also Appendix D).

TABLE The `class IntClass`

```
data member: int x; //variable to store the number
```

```
public IntClass()
  //constructor
  //When the object is instantiated, the value of x is initialized
  //to 0.
```

```
public IntClass(int num)
  //constructor
  //When the object is instantiated, the value of x is initialized
  //to num.
```

TABLE The **class** IntClass (continued)

```
public void setNum(int num)
  //Method to set the data member x.
  //The value of num is copied into x.

public int getNum()
  //Method to retrieve the value of x.
  //The value of x is returned.

public void addToNum(int num)
  //Method to update the value of x by adding the value of num.
  //x = x + num;

public void multiplyToNum(int num)
  //Method to update the value of x by multiplying by the value
  //of num.
  //x = x * num;

public int compareTo(int num)
  //Method to compare the value of x with the value of num.
  //Returns a value < 0 if x < num.
  //Returns 0 if x == num.
  //Returns a value > 0 if x > num.

public boolean equals(int num)
  //Method to compare x with num for equality.
  //Returns true if x == num; otherwise it returns false.

public String toString()
  //Method to return the value of x as a string.
```

Consider the following statements:

```
IntClass firstNum = new IntClass();         //Line 1
IntClass secondNum = new IntClass(5);       //Line 2
int num;                                    //Line 3
```

The statement in Line 1 creates the object `firstNum` and initializes it to `0`. The statement in Line 2 creates the object `secondNum` and initializes it to `5`. The statement in Line 3 declares `num` to be an **`int`** variable. Now consider the following statements:

```
firstNum.setNum(24);                        //Line 4
secondNum.addToNum(6);                      //Line 5
num = firstNum.getNum();                    //Line 6
```

The statement in Line 4 sets the value of `firstNum` (in fact, the value of the data member `x` of `firstNum`) to `24`. The statement in Line 5 updates the value of `secondNum` to `11` (the previous value `5` is updated by adding `6` to it). The statement in Line 6 retrieves the value of the object `firstNum` (the value of the data member `x`), and assigns it to `num`. After this statement executes, the value of `num` is `24`.

The following statements output the values of `firstNum` and `secondNum` (in fact, the values of their data members):

```
System.out.println("firstNum = " + firstNum);
System.out.println("secondNum = " + secondNum);
```

Because parameter passing is fundamental to any programming language, Examples 7-A and 7-B further illustrate this concept.

EXAMPLE 7-A: CALCULATE GRADE

Consider the following program. Given a course score (a value between `0` and `100`), it determines a student's course grade. This program to determine the course grade based on the course score has three methods: `main`, `getScore`, and `printGrade`. These three methods are described as follows:

1. `main`
   a. Get the course score.
   b. Print the course grade.
2. `getScore`
   a. Prompt the user for the input.
   b. Get the input.
   c. Print the course score.
3. `printGrade`
   a. Calculate the course grade.
   b. Print the course grade.

The complete program is as follows:

```
//Program: Compute the grade
//This program reads a course score and prints the
//associated course grade.

import java.util.*;

public class GradeProgram
{
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        DoubleClass courseScore = new DoubleClass();

        System.out.println("Line 1: Based on the course "
                         + "score, this program "
                         + "computes the course grade."); //Line 1
        getScore(courseScore);                          //Line 2
        printGrade(courseScore.getNum());               //Line 3
    }

    public static void getScore(DoubleClass score)
    {
        double s;                                       //Line 4

        System.out.print("Line 5: Enter the "
                       + "course score: ");             //Line 5
        s = console.nextDouble();                       //Line 6
        System.out.println();                           //Line 7

        score.setNum(s);                                //Line 8

        System.out.println("Line 9: The course "
                         + "score is " + s);            //Line 9
    }

    public static void printGrade(double testScore)
    {
        System.out.print("Line 10: Your grade for "
                         + "the course is ");           //Line 10

        if (testScore >= 90)                            //Line 11
            System.out.println("A");
        else if (testScore >= 80)
            System.out.println("B");
        else if (testScore >= 70)
            System.out.println("C");
        else if (testScore >= 60)
            System.out.println("D");
        else
            System.out.println("F");
    }
}
```

Sample Run: (In this sample run, the user input is shaded.)

```
Line 1: Based on the course score, this program computes the course grade.
Line 5: Enter the course score: 90.50

Line 9: The course score is 90.5
Line 10: Your grade for the course is A
```

This program works as follows. The program starts to execute at Line 1, which prints the first line of the output (see the Sample Run). The statement in Line 2 calls the method `getScore` with the actual parameter `courseScore` (a reference variable of the `DoubleClass` type declared in `main`). The formal parameter `score`, of the method `getScore`, is a reference variable of the `DoubleClass` type, which receives the value of `courseScore`. Thus, both `score` and `courseScore` point to the same object (see Figure 7-A1). Note that the method `getScore` also has a local variable `s`, declared in Line 4, of type **`double`**.

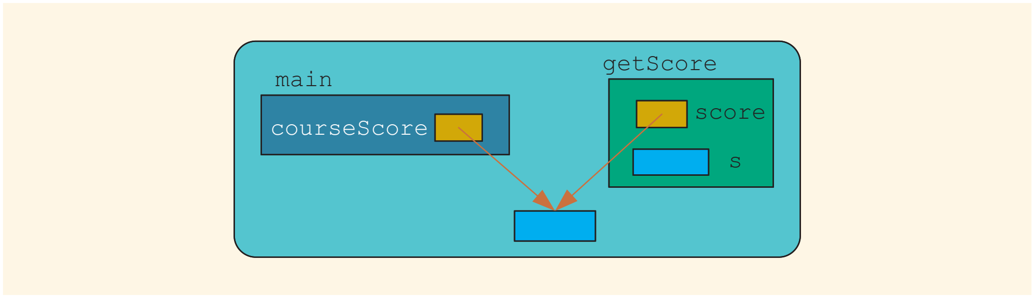

FIGURE 7-A1 Variable `courseScore` and the parameter `score`

Any changes that `score` makes to the object immediately change the value of the object to which `courseScore` points.

Control is then transferred to the method `getScore` and the statement in Line 5 executes. The statement in Line 5 prints the second line of output (see the Sample Run). This statement prompts the user to enter the course `score`. The statement in Line 6 reads and stores the value entered by the user (`90.50` in the Sample Run) in `s`. The statement in Line 8 copies the value of `s` into the object to which `score` points. Thus, at this point, the value of the object that `score` and `courseScore` point to is `90.50` (see Figure 7-A2).

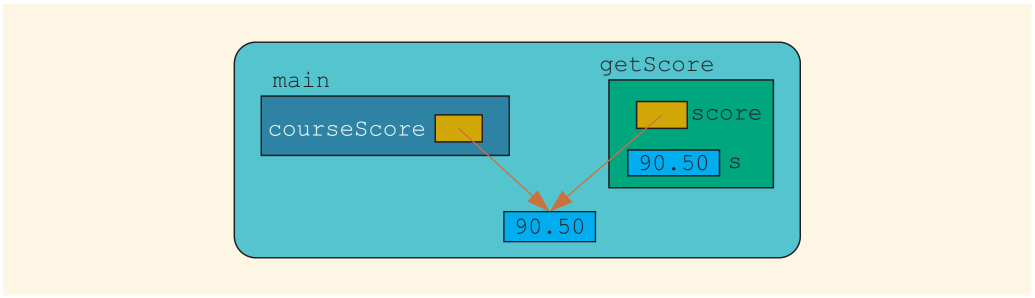

FIGURE 7-A2 Variable `courseScore` and the `parameter` score after the statement in Line 8 executes

Next the statement in Line 9 outputs the value of `s` as shown by the third line of the sample output. After the statement in Line 9 executes, control goes back to the method `main` (see Figure 7-A3).

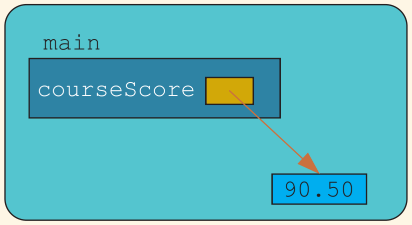

FIGURE 7-A3 Variable `courseScore` after the statement in Line 9 is executed and control goes back to `main`

The statement in Line 3 executes next. It is a method call to the method `printGrade` with the actual parameter `courseScore.getNum()`. In the expression `courseScore.getNum()`, the method `getNum` of the **`class`** `DoubleClass` returns the value of the object pointed to by `courseScore`, which is the course score. Now that the formal parameter `testScore` of the method `printScore` is of a primitive data type, the parameter `testScore` receives the course score. Thus, the value of `testScore` is `90.50` (see Figure 7-A4).

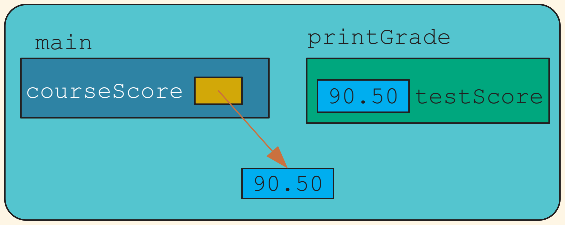

FIGURE 7-A4 Variable `courseScore` and the parameter `testScore`

Now the program executes the statement in Line 10, which outputs the fourth line (see the statement marked Line 10 in the Sample Run). The **`if...else`** statement in Line 11 determines and outputs the grade for the course. Notice that the output of the **`if...else`** statement is part of the fourth line of the output (see the Sample Run). After the **`if...else`** statement executes, control goes back to the method `main`. Because there are no more statements in method `main`, the program terminates.

In this program, the method `main` first calls the method `getScore` to get the course score from the user. The method `main` then calls the method `printGrade` to calculate and print the grade based on the course score. The course score is returned by the method `getScore`; later this course score is used by the method `printGrade`. Because the value retrieved by the `getScore` method is used later in the program, the method `getScore` must pass this value outside. Thus, the formal parameter that holds this value must be an object.

EXAMPLE 7-B: PRIMITIVE TYPES AS OBJECTS

```
public class Example_PrimitiveTypesAsObjects
{
    public static void main(String[] args)
    {
        int num1;                                         //Line 1

        IntClass num2 = new IntClass();                   //Line 2

        char ch;                                          //Line 3

        StringBuffer str;                                 //Line 4

        num1 = 10;                                        //Line 5
        num2.setNum(15);                                  //Line 6
        ch = 'A';                                         //Line 7
        str = new StringBuffer("Sunny");                  //Line 8

        System.out.println("Line 9: Inside main: "
                         + "num1 = " + num1
                         + ", num2 = " + num2.getNum()
                         + ", ch = " + ch
                         + ", and str = "
                         + str);                          //Line 9

        funcOne(num1, num2, ch, str);                     //Line 10

        System.out.println("Line 11: After funcOne: "
                         + "num1 = " + num1
                         + ", num2 = " + num2.getNum()
                         + ", ch = " + ch
                         + ", and str = "
                         + str);                          //Line 11
    }

  public static void funcOne(int a, IntClass b,
                             char v, StringBuffer pStr)
    {
        int num;                                               //Line 12
        int len;                                               //Line 13

        num = b.getNum();                                      //Line 14
        a++;                                                   //Line 15
        b.addToNum(12);                                        //Line 16
        v = 'B';                                               //Line 17

        len = pStr.length();                                   //Line 18
        pStr.delete(0, len);                                   //Line 19
        pStr.append("Warm");                                   //Line 20

        System.out.println("Line 21: Inside funcOne: \n"
                         + "         a = " + a
                         + ", b = " + b.getNum()
                         + ", v = " + v
                         + ", pStr = " + pStr
                         + ", len = " + len
                         + ", and num = " + num);              //Line 21
    }
}
```

Sample Run:

```
Line 9: Inside main: num1 = 10, num2 = 15, ch = A, and str = Sunny
Line 21: Inside funcOne:
         a = 11, b = 27, v = B, pStr = Warm, len = 5, and num = 15
Line 11: After funcOne: num1 = 10, num2 = 27, ch = A, and str = Warm
```

Let's walk through this program. The lines are numbered for easy reference, and the values of the variables are shown before and/or after each statement executes.

Just before the statement in Line 5 executes, memory is allocated only for the variables and objects of the method `main`. Note that `num1`, `ch`, and `str` are not initialized. The variable `num2` at Line 2 is declared, and the memory for the data is also allocated and initialized to `0`. This is because `num2` is a reference variable of `IntClass` type and the **`class`** `IntClass` automatically initializes the allocated memory space of the object to `0`. Just before the statement in Line 5 executes, the variables and objects are as shown in Figure 7-B1.

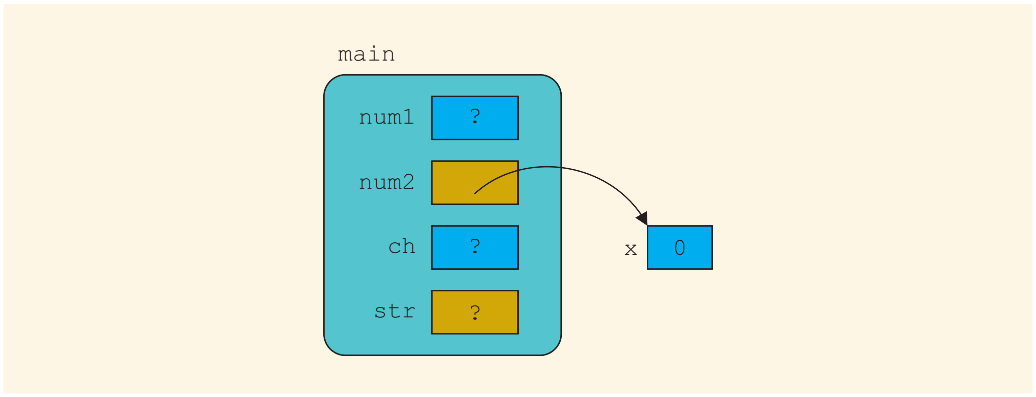

FIGURE 7-B1 Variables and objects just before the statement in Line 5 executes

The statement in Line 5 stores `10` into `num1`;, the statement in Line 6 stores `15` in the object pointed to by `num2`;, the statement in Line 7 stores the character `'A'` into `ch`;, and the statement in Line 8 allocates memory space to store the string `"Sunny"` into it and assigns this string to `str`. After the statement in Line 8 executes, the variables and objects of the method `main` are as shown in Figure 7-B2.

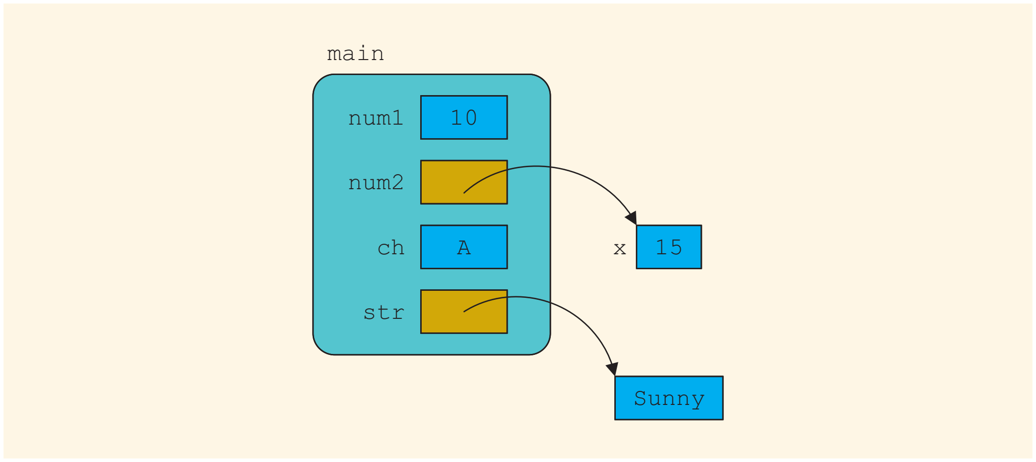

FIGURE 7-B2 Variables and objects after the statement in Line 8 executes

Line 9 produces the following output:

```
Line 9: Inside main: num1 = 10, num2 = 15, ch = A, and str = Sunny
```

The statement in Line 10 calls the method `funcOne`. The method `funcOne` has four parameters and two local variables. Memory for the parameters and the local variable of the method `funcOne` is allocated. The values of the actual parameters are copied into the corresponding formal parameters. Therefore, the value of `num1` is copied into `a`, the value of `num2` is copied into `b`, the value of `ch` is copied into `v`, and the value of `str` is copied into `pStr`. Now both the formal parameter `b` and `pStr` are reference variables, and so they point to the objects referred by their corresponding actual parameters.

Just before the statement in Line 14 executes, the variables and objects are as shown in Figure 7-B3.

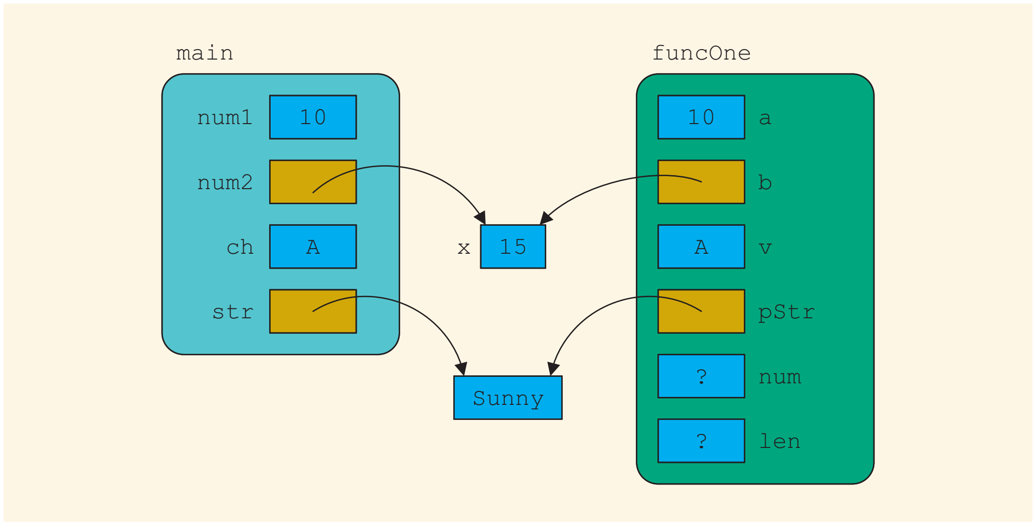

FIGURE 7-B3 Variables and objects just before the statement in Line 14 executes

The statement in Line 14 retrieves the value of `b` and stores it into `num`. After the statement in Line 14 executes, the variables and objects are as shown in Figure 7-B4.

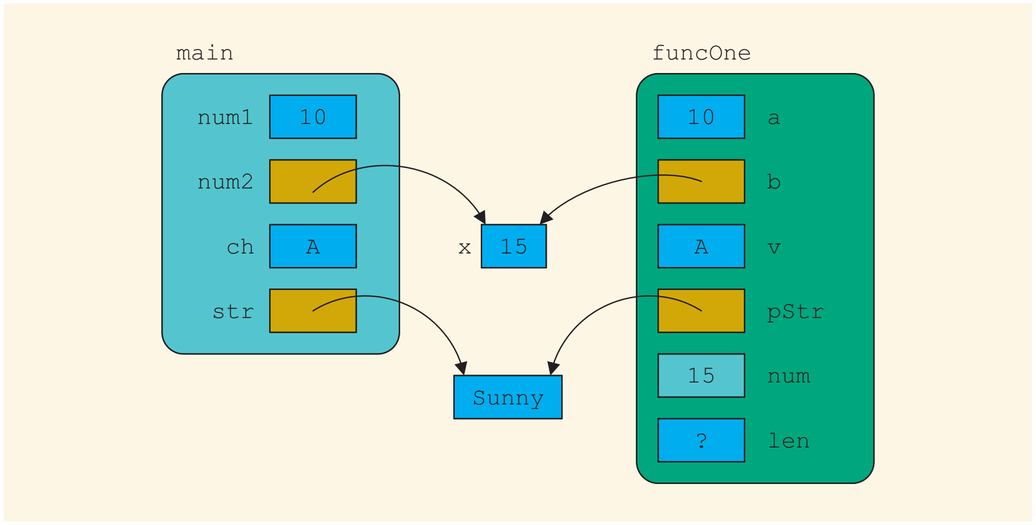

FIGURE 7-B4 Variables and objects after the statement, `num = b.getNum();`, in Line 14 executes

The statement in Line 15 increments the value of `a` by `1`. After the statement in Line 15 executes, the variables and objects are as shown in Figure 7-B5.

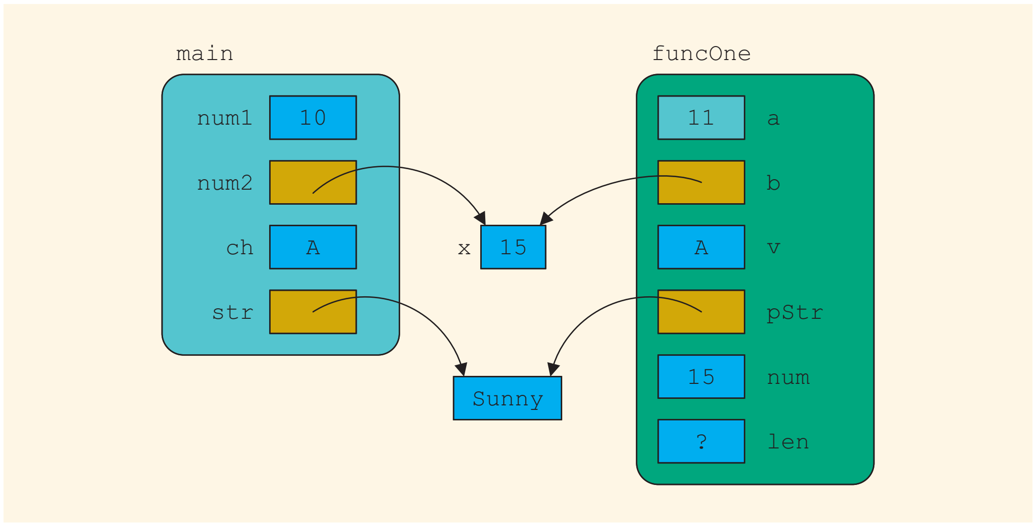

FIGURE 7-B5 Variables and objects after the statement, `a++;`, in Line 15 executes

The statement in Line 16 adds `12` to the previous value of the object referred by `b`. After the statement in Line 16 executes, the variables and objects are as shown in Figure 7-B6.

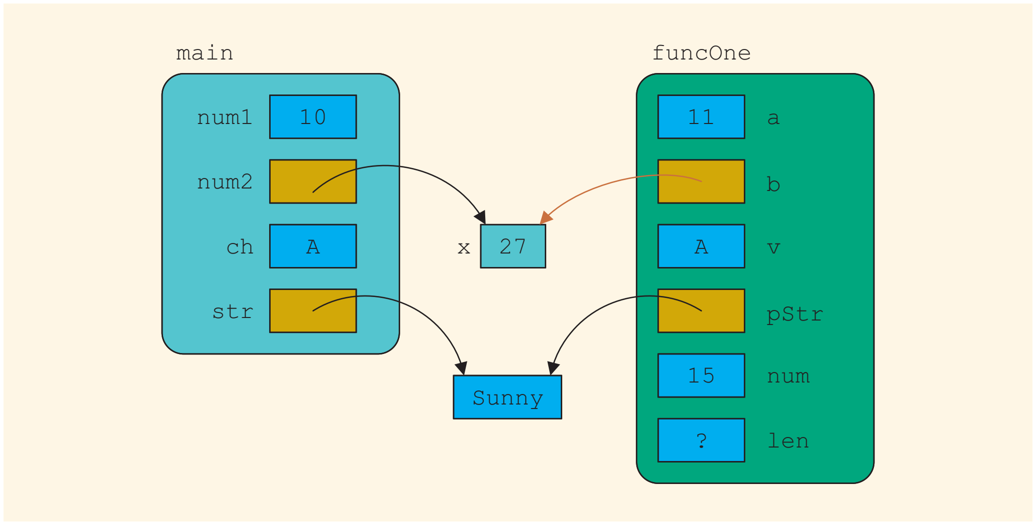

FIGURE 7-B6 Variables and objects after the statement, `b.addToNum(12);`, in Line 16 executes

The statement in Line 17 stores the character `'B'` into `v`. After the statement in Line 17 executes, the variables and objects are as shown in Figure 7-B7.

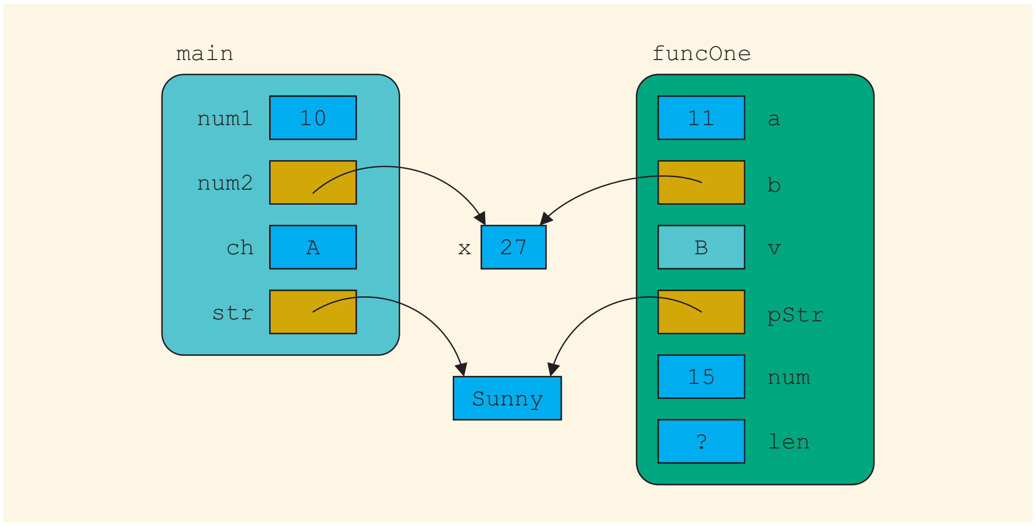

FIGURE 7-B7 Variables and objects after the statement, `v = 'B';`, in Line 17 executes

The statement in Line 18 stores the length of the string referred by `pStr` into `len`. After the statement in Line 18 executes, the variables and objects are as shown in Figure 7-B8. (Notice that the length of the string `"Sunny"` is `5`.)

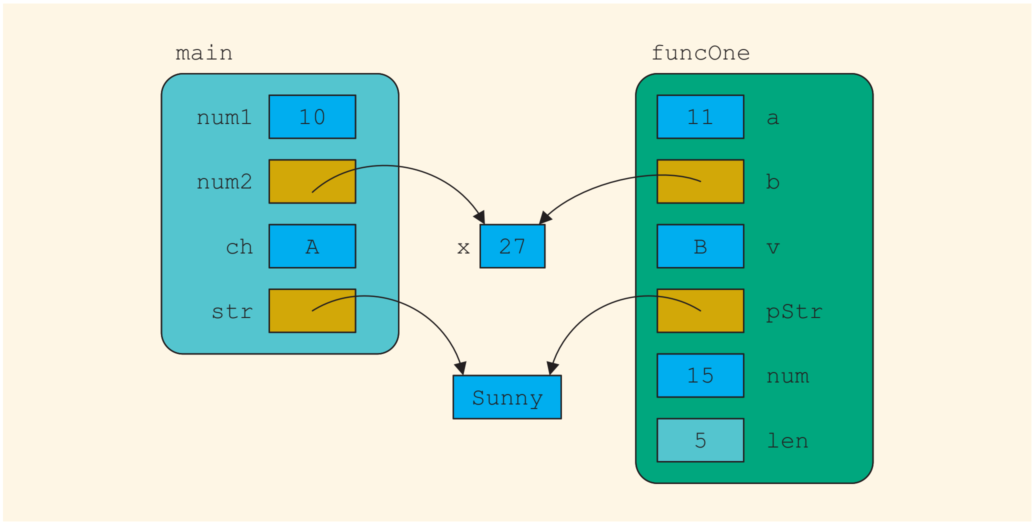

FIGURE 7-B8 Variables and objects after the statement, `len = pStr.length();`, in Line 18 executes

The statement in Line 19 uses the method `delete` of the `class StringBuffer` and deletes the string `"Sunny"`. After the statement in Line 19 executes, the variables and objects are as shown in Figure 7-B9.

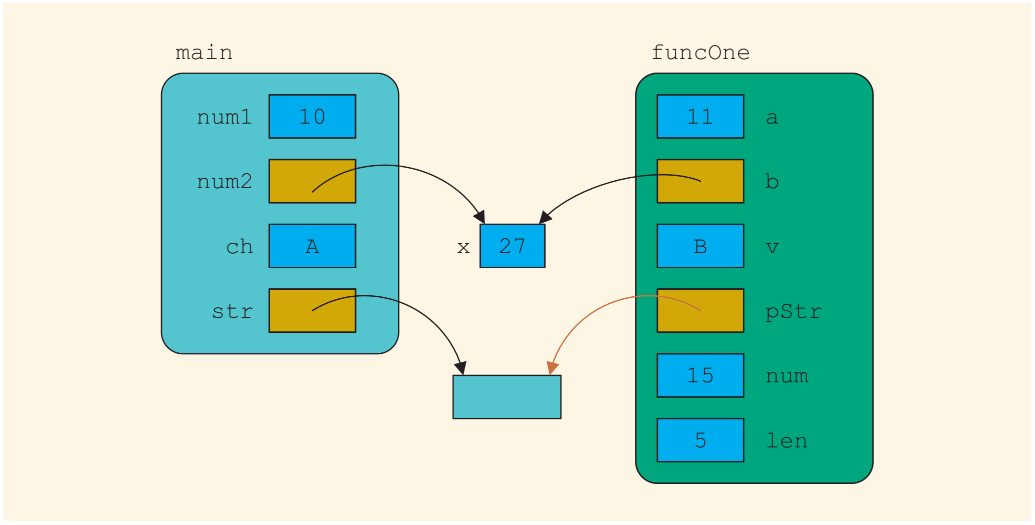

FIGURE 7-B9 Variables and objects after the statement, `pStr.delete(0, len);`, in Line 19 executes

The statement in Line 20 uses the method `append` of the **class** `StringBuffer` and appends the string `"Warm"` to the string referred by `pStr`. After the statement in Line 20 executes, the variables and objects are as shown in Figure 7-B10.

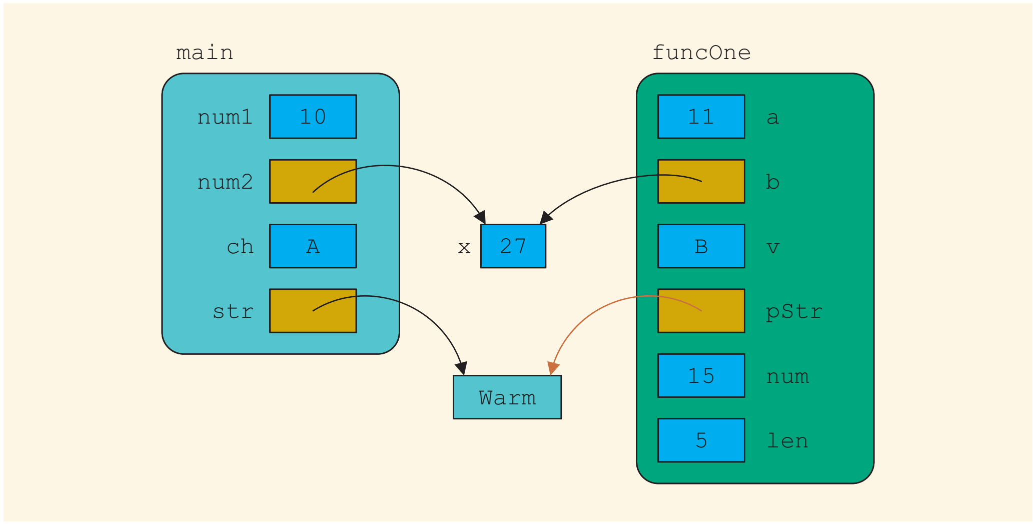

FIGURE 7-B10 Variables and objects after the statement, `pStr.append("Warm");`, in Line 20 executes

The statement in Line 21 produces the following output:

```
Line 21: Inside funcOne:
         a = 11, b = 27, v = B, pStr = Warm, len = 5, and num = 15
```

After the statement in Line 21 executes, control goes to Line 11. The memory allocated for the variables of method `funcOne` is deallocated. The values of the variables of the method `main` are as shown in Figure 7-B11.

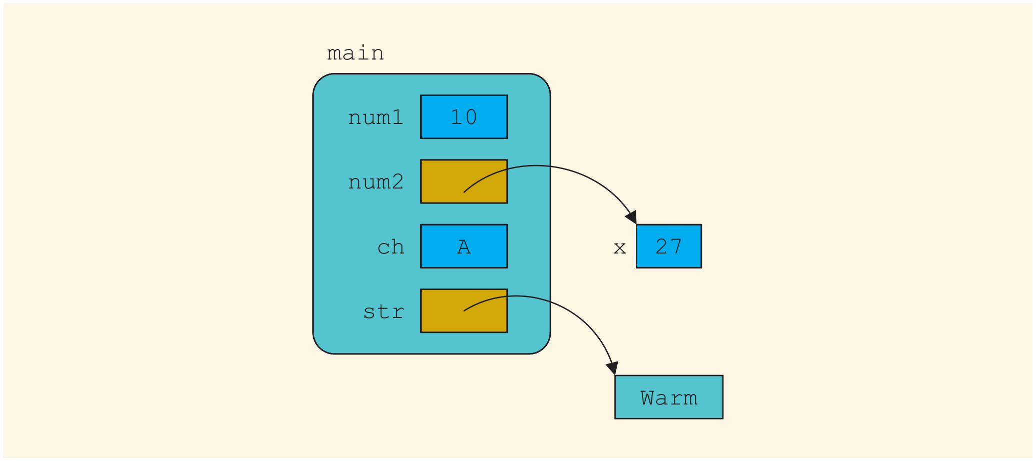

FIGURE 7-B11 Variables and objects of `main` when the control goes back to `main`

Line 11 produces the following output:

```
Line 11: After funcOne: num1 = 10, num2 = 27, ch = A, and str = Warm
```

After the statement in Line 11 executes, the program terminates.

EXERCISES

1. Assume the following input values:

```
7 3 6 4
2 6 3 5
```

Show the output of the following program:

```
import java.util.*;

public class Exercise1
{
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        int first, third;
        IntClass second, fourth;
        first = 3;
        second = new IntClass(4);
        third = 6;
        fourth = new IntClass(7);
        System.out.println(first + "  " + second.getNum()
                           + " " + third + "  "
                           + fourth.getNum());
        getData(first, second, third, fourth);
        System.out.println(first + "  " + second.getNum()
                           + "  " + third + "  "
                           + fourth.getNum());
        fourth.setNum(first * second.getNum() + third +
                      fourth.getNum());
        getData(third, fourth, first, second);
        System.out.println(first + "  " + second.getNum()
                           + "  " + third + "  "
                           + fourth.getNum());
    }

    public static void getData(int a, IntClass b, int c,
                               IntClass d)
    {
        a = console.nextInt();
        b.setNum(console.nextInt());
        c = console.nextInt();
        d.setNum(console.nextInt());

        b.setNum(a * b.getNum() - c);
        d.setNum(b.getNum() + d.getNum());
    }
}
```

2. Consider the following program. What is its exact output? Show the values of the variables after each line executes, as in Example 7-B.

```
public class Exercise2
{
    public static void main(String[] args)
    {
        int num1;
        IntClass num2;

        num1 = 10;                                               //Line 1
        num2 = new IntClass(20);                                 //Line 2

        System.out.println("Line 3: In main: num1 = "
                          + num1 + ", num2 = "
                          + num2.getNum());                      //Line 3

        funcOne(num1, num2);                                     //Line 4

        System.out.println("Line 5: In main after funcOne: "
                         + "num1 = " + num1 + ", num2 = "
                         + num2.getNum());                       //Line 5
    }
    public static void funcOne(int a, IntClass b)
    {
        int x;
        int z;

        x = b.getNum();                                 //Line 6

        z = a + x;                                      //Line 7

        System.out.println("Line 8: In funcOne: a = " + a
                          + ", b = " + b.getNum()
                          + ", x = " + x
                           + ", and z = " + z);         //Line 8

        x = x + 5;                                      //Line 9

        System.out.println("Line 10: In funcOne: a = " + a
                          + ", b = " + b.getNum()
                          + ", x = " + x
                          + ", and z = " + z);          //Line 10

        a = a + 8;                                      //Line 11

        b.setNum(a + x + z);                            //Line 12

        System.out.println("Line 13: In funcOne: a = " + a
                          + ", b = " + b.getNum()
                          + ", x = " + x
                          + ", and z = " + z);          //Line 13
    }
}
```

PROGRAMMING EXERCISES

1. Consider the following program segment (with the method main):

```
public class Ch7_Web_PrExercise1
{
    public static void main(String[] arg)
    {
        IntClass x, y;
        CharClass z;

        double rate;
        double hours;
        double amount;
        .
        .
        .
    }
}
```

Write the following definitions:

- a. Write the definition of the method `initialize` that initializes `x` and `y` to `0` and `z` to the blank character.
- b. Write the definition of the method `getHoursRate` that prompts the user to input the hours worked and rate per hour to initialize the variables `hours` and `rate` of the method `main`.
- c. Write the definition of the value-returning method `payCheck` that calculates and returns the amount to be paid to an employee, based on the hours worked and rate per hour. The hours worked and rate per hour are stored in the variables `hours` and `rate`, respectively, of the method `main`. The formula for calculating the amount to be paid is as follows: for the first 40 hours, the rate is the given rate; for hours over 40, the rate is 1.5 times the given rate.
- d. Write the definition of the method `printCheck` that prints the hours worked, rate per hour, and the amount due.
- e. Write the definition of the method `funcOne` that prompts the user to input a number. The method then changes the value of `x` to 2 times the old value of `x` plus the value of `y` minus the value entered by the user.
- f. Write the definition of the method `nextChar` that sets the value of `z` to the next character stored in `z`.
- g. Write the definition of a method `main` that tests each of these methods. Also write a complete program to test the methods of (a)–(f).

2. The method `printGrade` of this chapter's Example A, is written as a **`void`** method to compute and output the course grade. The course score is passed as a parameter to the method `printGrade`. Rewrite the method `printGrade` as a value-returning method so that it computes and returns the course grade. (The course grade must be output in the method `main`.) Also, change the name of the method to `calculateGrade`.
3. In this exercise, you are to rewrite the Programming Example Classify Numbers (from Chapter 5). As written, the program inputs data from the standard input device (keyboard) and outputs results on the standard output device (screen). The program can process only 20 numbers. Rewrite the program to incorporate the following requirements:
   - a. Data to the program is input from a file of unspecified length; that is, the program does not know in advance how many numbers are in the file.
   - b. Save the output of the program in a file.
   - c. Write the method `getNumber` so that it reads a number from the input file (opened in the method `main`), outputs the number to the output file (opened in the method `main`), and sends the number read to the method `main`. Print only 10 numbers per line.

   d. Have the program find the sum and average of the numbers.
   e. Write the method `printResult` so that it outputs the final results to the output file (opened in the method `main`). Other than outputting the appropriate counts, the definition of the method `printResult` should also output the sum and average of the numbers.

4. Rewrite the program developed in Programming Exercise 17 in Chapter 5, so that the method `main` is merely a collection of method calls. Your program should use the following methods:
   a. Method `initialize`: This method initializes variables such as `countFemale`, `countMale`, `sumFemaleGPA`, and `sumMaleGPA`.
   b. Method `sumGrades`: This method finds the sum of female and male students' GPAs.
   c. Method `averageGrade`: This method finds the average GPA for female and male students.
   d. Method `printResults`: This method outputs the relevant results.

   Use appropriate parameters to pass information in and out of methods.

5. Write a program to process text files. The program should read a text file and output the data in the file as is. The program should also output the number of words, lines, and paragraphs.

   You must write and use the following methods:
   a. `initialize`: This method initializes all variables of the method `main`.
   b. `processBlank`: This method reads and writes blanks. Whenever it hits a nonblank (except whitespace characters), it increments the number of words in a line (this number is set back to zero in the method `updateCount`). The method exits after processing blanks.
   c. `copyText`: This method reads and writes nonblank characters. Whenever it hits a blank, it exits.
   d. `updateCount`: This method takes place at the end of each line. It increments the number of lines and sets the number of words on a line back to zero. If there are no words in a line, it increments the number of paragraphs. One blank line (between paragraphs) is used to distinguish paragraphs and should not be counted with the number of lines. This method also updates the total word count.
   e. `printTotal`: This method outputs the number of words, number of lines, and number of paragraphs.

   Your program should read data from a file and send output to a file. Use appropriate parameters to pass values in and out of methods. Test your program using the method `main` that looks like the following:

```
public static void main(String[] args)
{
    variables declaration
    open files

    read a character

    while (not end of file)
    {
        while (not end of line)
        {
            processBlank(parameters);
            copyText(parameters);
        }

        updateCount(parameters);
        read a character;
        ...
    }

    printTotal(parameters);
    close files;
}
```

Because the program needs to do a character count, it should read the input file character-by-character. The program should also count the number of lines. Therefore, while reading data from the input file, the program must capture the newline character. The `Scanner` **class** does not contain any method that can only read the next character in the input stream unless the character is delimited by whitespace characters such as blanks. Using the `Scanner` **class**, either the program should read the entire line or the newline character will be ignored.

To simplify the reading of the input file character-by-character, you can use the Java **class** `FileReader`. (In Chapter 3, we introduced this class to create and initialize a `Scanner` object to the input source.) The **class** `FileReader` contains the method `read` that returns the integer value of the next character. For example, if the next input character is `'A'`, then the method `read` returns `65`. We can use the cast operator to change the value `65` to the character `'A'`. Note that the method `read` *does not* skip whitespace characters. The method `read` returns `-1` when the end of the input file has been reached. You can therefore use the value returned by the method read to determine whether the end of the input file is reached.

Consider the following statement:

```
FileReader inputStream = new FileReader("text.txt");
```

This statement creates the `FileReader` object `inputStream` and initializes it to the input file `text.txt`. If `nextChar` is a **char** variable, then the following statement reads and stores the next character, from the input file, into `nextChar`:

```
ch = (char) inputStream.read();
```