03 Working with Arrays, Vectors, strings, and Wrapper Classes

IN THIS CHAPTER

3.1 Introduction to Computers

As a Java programmer, you may need to create programs to process information of similar types. For example, you have to create a program to store roll numbers of 25 students in a class. All the roll numbers are of numeric type; therefore, they should be stored in integer data type. Now, you declare 25 variables to store 25 roll numbers. Declaring and remembering such a large number of variables are a very tedious task. To get rid of this kind of hectic work, Java provides `arrays`. Using arrays, you can use only a single variable to store large numbers of elements of same data type. However, you cannot delete the elements stored in an array. You have to define the size of the array while creating it. Therefore, you can only store total number of the elements equal to the size of the array. You cannot add elements beyond the size of the array. However, in case, you have to add more elements in an array then you have to create another array. For managing two or more arrays, Java provides the `Vector` class, which is used to create an object similar to an array. The object created using the `Vector` class is dynamic that is the object size is automatically increased during runtime to store the additional elements.

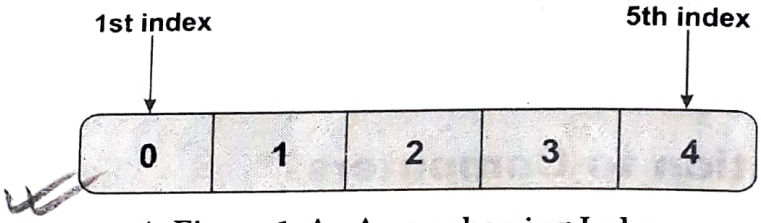
In Java, primitive values such as `int`, `char`, and `byte` cannot be put into a collection because a collection can hold only object references. Therefore, to put primitive values into a collection, they must be converted into objects. Java provides wrapper classes to convert primitive values to objects. Apart from wrapper classes, Java also provides the `String` and `StringBuffer` classes that are used to handle text strings.

In this chapter, we study various types of arrays available in Java. We also study about Vector class. Apart from that, we cover the Wrapper classes and learn how they help in converting primitive values into objects. Towards the end of the chapter, we study the `String` and `StringBuffer` classes along with various methods to work with text strings in Java. Let's move ahead in this chapter with arrays.

3.2 Using Arrays in Java

An array is a collection of data storage locations, each of which holds the same type of data. There can be a situation where you need to store similar type of values for a large number of data items. For example, to store telephone numbers of 50 persons, normally you have to declare 50 variables each storing a number in it, which is not a correct approach and also a very tedious and time consuming task.

With arrays, we can declare a single variable that is capable of storing all the 50 numbers. The advantage of using arrays is that they simplify programming by replacing a number of statements with just one or two statements. However, the size of an array cannot be modified. All the values stored in an array are considered as elements. All the elements in an array are stored at a specific index number. An index number in an array starts with 0 and continues up to the size of an array. For example, if you have created an array to store 5 elements then index numbers will range from 0-4 that is the 5th element is accessed by the index number 4. See Figure 1, which is explaining indexes in an array:



▲ **Figure 1: An Array showing Indexes**

In Figure 1, you can see that the 0th element is showing the first index while the 4th element is showing the 5th index. Let's discuss arrays under the following heads:

- One-Dimensional Array
- Multi-dimensional Array
- Using Array of Objects
- Using the System.arrayCopy() Method

Let's start studying arrays in the given sequence.

■ Using One-Dimensional Array

A one-dimensional array is a collection of data storage locations, with one dimension. The number of indexes associated with any array is its dimension. In one dimensional array, the array elements are represented with single index value. Here, we discuss the following broad-level steps to use a one-dimensional array:

- Declaring an One-Dimensional Array
- Creating an One-Dimensional Array
- Initializing an One-Dimensional Array
- Developing a Program Using One-Dimensional Array

Now, let's discuss these steps in the following section.

➧ Declaring a One-Dimensional Array

The first step to start working with an array is to declare an array. In Java, an array is declared in the same way as a data variable is declared. The syntax to declare a one-dimensional array is:

```
data_type var_name [ ];
```

In the preceding syntax:

- **data_type** is the data type of an array, which specifies the type of elements that will be stored in an array.
- **var_name** is the name of the array.

Another syntax to declare a one-dimensional array is:

```
data_type [ ] var_name;
```

See the following examples in which arrays of name roll_no and tele_no of data type int are declared:

```
int roll_no [ ];
int [ ] tele_no;
```

Both arrays given in preceding examples are declared correctly according to the Java syntaxes. Declaring an array does not mean to create an array. To create an array, memory must be allocated to it. Until memory is not allocated to an array, the `null` value is assigned to the declared array.

Creating a One-Dimensional Array

Once you have declared a one-dimensional array, the next step is to actually create that array by allocating memory to it. The new operator is used to allocate memory to an array. The syntax to create an array is as follows:

```
var_name = new data_type [size];
```

In the preceding syntax:

- **var_name** is the name of the array declared in the preceding section
- **new** is the operator, which is used to allocate memory to var_name.
- **data_type** is the data type of the elements that will be stored in the array.
- **size** is the total number of elements that will be stored in the array.

Total memory that will be allocated to an array depends on the data_type of the array elements and the size of the array. The following example creates an array of size 10:

```
roll_no = new int [10];
```

According to the preceding example, the total number of the elements that roll_no can be stored are 10. The total amount of the memory allocated to roll_no is 40 bytes. The array, roll_no is allocated 40 bytes because total numbers of elements are 10 and have data type int. An element of type int takes 4 bytes, therefore, 4 * 10 = 40.

You can also combine the array's declaration and creation in a single step, as shown here:

```
int roll_no [ ] = new int [10];
```

Combining declaration and creation in a single step saves the time. Now, when the array roll_no is created, it's time to initialize it.

Initializing a One-Dimensional Array

Adding values in an array is known as initialization. The syntax to initialize an array is as follows:

```
var_name [ index_no ] = value;
```

In the preceding syntax:

- **var_name** is the array name.
- **index_no** is the array index where the value is to be stored.
- **value** can be any number or character, which is stored at the index_no.

See the following example to initialize an array:

```
roll_no [0]=121;
roll_no [1]=122;
roll_no [2]=123;
roll_no [3]=124;
roll_no [4]=125;
roll_no [5]=126;
roll_no [6]=127;
roll_no [7]=128;
roll_no [8]=129;
roll_no [9]=130;
```

You can also initialize an array at the time of declaration. If you initialize an array at the time of declaration then you do not need to use the new keyword to assign memory to array elements. You also do not need to specify size of an array. Syntax to initialize an array at the time of declaration is as follows:

```
data_type var_name [ ] = { array_element1, array_elements2, array_elementn};
```

In the preceding syntax:

- **data_type** is the data type of an array, which specifies the type of elements that will be stored in an array.
- **var_name** is the name of the array.
- **array_element1** is the first array element, which is to be stored in the array. You can store n numbers of elements in an array. Commas must separate all the array elements.

While initializing an array at the time of declaration, you do not need to specify the array size. This is because, the JVM at runtime itself calculates the array size based on the total number of the elements added in the array. Therefore, the JVM also internally assigns memory to the array on the basis of the calculated array size. The following example initializes an array, roll_no at the time of array declaration:

```
int roll_no [ ] = { 121, 122, 123, 124, 125, 126, 127, 128, 129, 130};
```

You can also assign an array object to another array object, as follows:

```
int roll_no [ ]= {121, 122, 123};
int roll_no1 [ ];
roll_no1 = roll_no;
```

Now, when an array is initialized, you can start working with it.

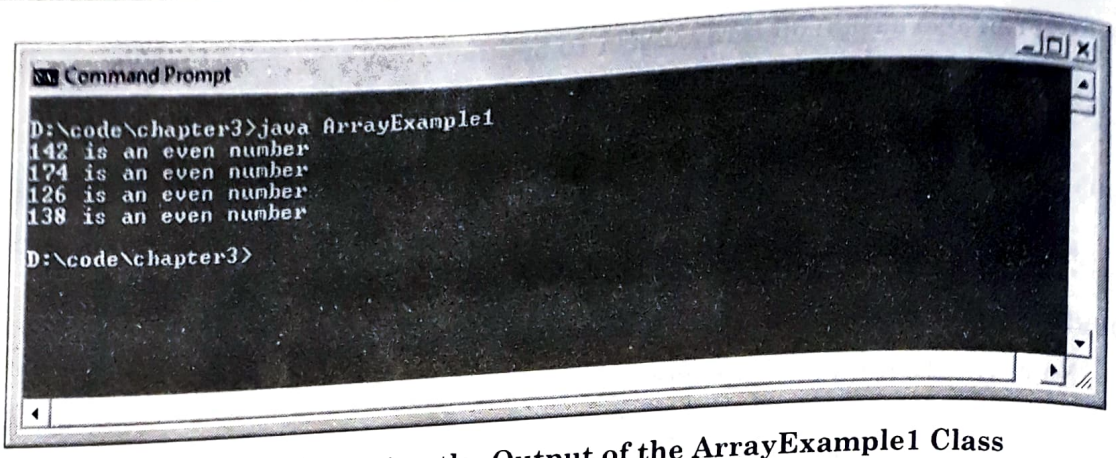
Developing a Program Using One-Dimensional Array

Now, we create a Java program to declare, create, and initialize an array. After initializing an array, you can display array elements and can also perform operations such as addition, subtraction on them. Listing 1 segregates and displays even values from the elements stored in an array (you can find the ArrayExample1.java file on the CD in the code\chapter3 folder):

▶ **Listing 1: Program to Segregate and Display Even Values**

```
public class ArrayExample1
{
 public static void main(String args[])
 {
     int x;
     int even[];
     even = new int[10];
     even [0]=121;
     even [1]=142;
     even [2]=145;
     even [3]=149;
     even [4]=163;
     even [5]=174;
     even [6]=126;
     even [7]=129;
     even [8]=138;
     even [9]=145;
     for(x=0;x<even.length;x++)
     {
         if(even[x]%2==0) {
             System.out.println(even[x]+ " is an even number");
         }
     }
 }
}
```

In Listing 1, a one-dimensional array of `int` type called `even` is declared to group simple data types. The array is then initialized with 10 numeric values. These values are accessed using for loop where each value stored in the array is accessed using their respective index numbers. Accessed values are then checked for even numbers and are displayed one by one. In for loop, we used the `length` variable to execute the loop up to the allocated size. The length variable is a built-in Java variable, which stores an array size. The output of Listing 1 is shown in Figure 2:



▲ **Figure 2: Displaying the Output of the ArrayExample1 Class**

This is all about one-dimensional array. Now, we study about Multi-dimensional array.

TEST YOUR KNOWLEDGE

Q1. **Write a program to create an array to store 5 integer values. Also initialize the array with 5 random numbers and display the array elements in reverse order.**

Ans.

```
class TestYourKnowledge1
{
 public static void main(String args[])
 {
     int x[]={8,7,2,9,6};
     int y=x.length;
     for(int j=y-1; j>=0; j--)

     {
         System.out.print(x[j]);
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge1.java
D:\code\chapter3>java TestYourKnowledge1
```

Output:

```
69278
```

Q2. **Write a program to create an array to store 10 integer values. Also initialize the array with 10 random numbers, sort the array, and display the array elements.**

Ans.

```
class TestYourKnowledge2
{
 public static void main(String args[])
 {
     int x[]={8,7,2,9,6,1,3,78,25,15};
     int temp;
     int y=x.length;
     for(int j=0; j<y; j++)
```

```
        {
            for(int m=1; m<(y-j); m++)
            {
                if(x[m-1]>x[m])
                {
                    temp = x[m-1];
                    x[m-1] = x[m];
                    x[m] = temp;
                }
            }
        }
        for(int k=0; k<y;k++)
        {
            System.out.print(x[k] + " ");
        }
    }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge2.java
D:\code\chapter3>java TestYourKnowledge2
```

Output:

```
1 2 3 6 7 8 9 15 25 78
```

Q3. Write a program to create an array to store 10 integer values. Also initialize the array with 10 random numbers and display all even values.

Ans.

```
class TestYourKnowledge3
{
  public static void main(String args[])
  {
      int x[]={2, 4, 5, 7, 9, 11, 48,  52, 69, 25};
      for(int j=0; j<x.length;j++)
      {
          if(x[j]%2==0)
          {
              System.out.print(x[j]  + " ");
          }
      }
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge3.java
D:\code\chapter3>java TestYourKnowledge3
```

Output:

```
2 4 48 52
```

Q4. Write a program to create an integer array containing 10 values. Then print all the prime numbers contained by the array. A prime number is a number, which can either be divided by 1 or by itself.

Ans.

```
class TestYourKnowledge4
{
 public static void main(String args[])
 {
     int num[] ={2,4,5,6,7,9,11,13,14,16};
     int i,j;
     for(j=0;j<num.length;j++)
     {
         for(i=2; i<=num[j]; i++)
         {
             int n = num[j]%i;
             if(n == 0 )
             {
             break;
             }
         }

         if(i == num[j])
         {
             System.out.print(" " + num[j]);
         }
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge4.java
D:\code\chapter3>java TestYourKnowledge4
```

Output:

```
2 5 7 11 13
```

Q5. Write a program to create a character array and display the ASCII equivalent of the characters stored in that array.

Ans.

```
class TestYourKnowledge5
{
 public static void main(String args[])
 {
     char c[]={'#', '%', ' ', '8', 'u'};
     for(int x=0; x<c.length; x++)
     {
         int i = (int)c[x];
         System.out.println(i + " " );
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge5.java
D:\code\chapter3>java TestYourKnowledge5
```

Output:

```
35
37
32
56
117
```

Q6. Write a program to find the Mode of given numbers.

Ans.

```
class TestYourKnowledge6
{
 public static void main(String args [])
 {
     int N,K,I,BIG;
     int F[]=new int[20];
     int A[]={25,24,26,25,26,26,26,24,26,27,27,25,26,27,27};
     N=A.length;
     for(K=0;K<=N-1;K++)
     {
         F[K]=0;
         for(I=0;I<=N-1;I++)
         {
             if(A[K]==A[I])
             ++F[K];
         }
     }
     BIG=F[0];
     int LOC=0;
     for(I=1;I<=N-1;I++)
     {
         if (BIG<F[I])
         {
             BIG=F[I];
             LOC=I;
             System.out.print("MODE = " +A[LOC]);
         }
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge6.java
D:\code\chapter3>java TestYourKnowledge6
```

Output:

```
Mode=26
```

Q7. Write a program to find the first and the second biggest number from given numbers.

Ans.

```
// biggest first & second no.
import java.lang.Math;
class TestYourKnowledge7
{
 public static void main (String args[])
 {
     int A[]={16,56,34,98,46,34,76,87};
     int BIG1=A[0];
     int n=A.length;
     for (int I=1; I<n-1; I++)
     if (BIG1<A[I])
     BIG1=A[I];
     System.out.println("Biggest No = " +BIG1);
     int I=0;
     while (A[I]==BIG1)
     ++I;
     int BIG2=A[I];
     for (I=0; I<=n-1; I++)
     if ((BIG2<A[I])&&(A[I]!=BIG1))
     BIG2=A[I];
     System.out.println("Second Biggest = " +BIG2);
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge7.java
D:\code\chapter3>java TestYourKnowledge7
```

Output:

```
Biggest No = 97
Second Biggest = 87
```

Q8. Write a program to find the sum and average of even numbers in a given array.

Ans.

```
class TestYourKnowledge8
{
 public static void main(String args[])
 {
     int count=0,N,I,SUM=0;
     float AVG=0.0f;
     int A[] = {10,22,11,30,49,31,42,27,39,61,93,76};
     N=A.length;
     for (I=0;I<N;I++)
     if (A[I]%2==0)
     {
         SUM+=A[I]; ++count;
         AVG=(float) (SUM/count);
     }
     System.out.println("SUM="+SUM);
     System.out.println("AVG="+AVG);
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge8.java
D:\code\chapter3>java TestYourKnowledge8
```

Output:

```
SUM=180
AVG=36.0
```

Q9. Write a program to ignore elements from a given array.

Ans.

```
class TestYourKnowledge9
{
 public static void main(String args[])
 {
     int A[]={0,14,26,39,11,46,36,59,18};
     int N= A.length-1;
     System.out.println("Array before deletion of elements is:   ");
     for(int k=0;k<=N;k++)
     {
         System.out.print(" " + A[k]);
     }
     int I=0,loc;
     while(A[++I]!=11);
     loc=I;
     for(I=loc;I<=N-1;I++)
     A[I]=A[I+1];
     N=N-1;
     System.out.println("   ");
     System.out.println("Array after deletion of elements is:   ");
     for(I=1;I<=N;I++)
     System.out.print("  "+A[I]);
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge9.java
D:\code\chapter3>java TestYourKnowledge9
```

Output:

```
Array before deletion of elements is:
0 14 26 39 11 46 36 59 18
Array after deletion of elements is:
14 26 39 46 36 59 18
```

Q10. Write a program to find the sum of odd numbers in an array.

Ans.

```
class TestYourKnowledge10
{
 public static void main (String args[])
 {
```

```
        int count=0,N,I,SUM=0;
        int A[]={10,22,11,36,49,31,42,27,39,27,39,61,93,76};
        N=A.length;
        for (I=0;I<=N-1;I++)
        {
            if (A[I]%2!=0)
            {
                SUM+=A[I];
                ++count;
            }

        }

        System.out.println("SUM="+ SUM);
    }

}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge10.java
D:\code\chapter3>java TestYourKnowledge10
```

Output:

```
SUM=186
Average=37.0
```

Q11. Write a program to know that whether an array is arranged in decreasing order or not.

Ans.

```
// To verify the array whether it is arranged in decreasing order or
 not
// class TestYourKnowledge11
{
 static boolean order(int n,int a[])
 {
    boolean flag1=true, flag=true ;
    for(int i=0;i<=n-2;i++)
    {
        if(a[i]<a[i+1]) flag1=false;
    }

    return(flag1);
}

public static void main(String args[])
{
    int n=5;
    int a[]={75,65,55,45,35};
    if (order(n,a))// when flag1=true;
    System.out.println("Array is arranged in order");
    else
    System.out.println("Array is not arranged in order");
}
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge11.java
D:\code\chapter3>java TestYourKnowledge11
```

Output:

```
Array is arranged in order
```

Q12. Write a program to sort an array in ascending order.

Ans.

```
class TestYourKnowledge12
{
  public static void main(String args [])
  {
      int N,I,J,TEMP,LOC;
      int A[]={52,36,96,58,47,41,67,49,16,28};
      N=A.length;
      for (I=0;I<=N-1;I++)
      {
          int small=A[I];
          LOC=I;
          for (J=I+1;J<=N-1;J++)
          if (small>A[J])
          {
              small=A[J];
              LOC=J;
          }
          TEMP=A[I];
          A[I]=A[LOC];
          A[LOC]=TEMP;
      }
      for (I=0;I<=N-1;I++)
      System.out.print(" "+A[I]);
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge12.java
D:\code\chapter3>java TestYourKnowledge12
```

Output:

```
16 28 36 41 47 49 52 58 67 96
```

Q13. Write a program to find the Median of the given data.

Ans.

```
class TestYourKnowledge13
{
  public static void main(String args [])
  {
      int N,K,I,TEMP,LOC,small;
      float MEDIAN=0.0f;
```

```
            int A[]={52,36,96,58,47};
            N=A.length;
            for (K=0;K<=N-1;K++)
            {
                small=A[K];
                LOC=K;
                for (I=K+1;I<=N-1;I++)
                if (small>A[I])
                {
                    small=A[I];
                    LOC=I;
                }
                TEMP=A[K];
                A[K]=A[LOC];
                A[LOC]=TEMP;
                int NN=(N-1)/2;
                if(NN%2==0)
                MEDIAN=A[NN];
            }

            System.out.println("MEDIAN="+MEDIAN);
        }
    }
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge13.java
D:\code\chapter3>java TestYourKnowledge13
```

Output:

```
MEDIAN=52.0
```

Q14. Write a program to find the SD of the given data.

Ans.

```
import java.lang.System;
class TestYourKnowledge14
{

    public static void main(String args[])
    {
        int I,N,KI,TEMP,LOC,BIG;
        int A[]={52,59,69,86,74,38,91};
        N=A.length;
        Float SUM=0.0f;
        for (I=0;I<=N-1;I++)
        SUM+=A[I];
        float MEAN=(float) SUM/N;
        float sigma=0;
        for (I=0;I<=N-1;I++)
        sigma+=(MEAN-A[I])*(MEAN-A[I]);
        System.out.print("STD.DEV. ="+(sigma/N));
    }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge14.java
D:\code\chapter3>java TestYourKnowledge14
```

Output:

```
STD.DEV. =302.85715
```

Q15. Write a program to find the mode (most frequently occurring item) of set of numbers.

Ans.

```
class TestYourKnowledge15
{
 public static void main (String args[])
 {
     int A[] = {0,2,5,2,5,8,2,5,8,8,6,3,7,8,5,5,9,2,7,2,8,6};
     int N = A.length-1;
     int loc= 0,BIG=0;
     for (int i = 1 ; i <= N; i++)
     {
         int count= 0;
         for (int j = 1;j<=N;j++)
         if (A[i] == A[j]) count++;
         if (BIG  <count)
         {BIG = count;
         loc = i;
      }
}
System.out.println("Mode is " + A[loc]);
}
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge15.java
D:\code\chapter3>java TestYourKnowledge15
```

Output:

```
Mode is 2
```

Using Multi-dimensional Array / Two-dimension Array

So far, we have seen only `one-dimensional` arrays where only a single value can be allocated but there can be a case when more values are needed to represent the elements of an array. In that case, `multi-dimensional` array can be used to represent the data.

A multi-dimensional array is also known as array of arrays. The process to declare, create, initialize, and use a multi-dimensional array will be discussed in the following headings:

- Declaring a Multi-dimensional Array
- Creating a Multi-dimensional Array
- Initializing an Multi-dimensional Array

- Declaring an Uneven Multi-dimensional Array
- Developing a Program Using Multi-dimensional Array

Now, let's discuss these processes in the following sections.

Declaring a Multi-dimensional Array

Declaring a multi-dimensional array is the first step to start working with a multi-dimensional array. The syntax to declare a multi-dimensional array is:

```
data_type var_name [ ] [ ];
```

In the preceding syntax:

- **data_type** is the data type of an array, which specifies the type of elements that will be stored in an array.
- **var_name** is the name of the array.
- **[ ]** is the first/left bracket pair, which represents the row numbers.
- **[ ]** is the second/right bracket pair, which represents the column numbers.

Another syntax to declare a multi-Dimensional array is:

```
data_type [ ] [ ] var_name;
```

See the following examples in which arrays of name `even_no` and `odd_no` of data type `int` are declared:

```
int even_no [ ] [ ];
int [ ] [ ] odd_no;
```

Both arrays in given preceding examples are declared correctly according to the Java syntaxes to declare multi-dimensional arrays. Now, when a multi-dimensional array is declared, you can move ahead to create a multi-dimensional array.

Creating a Multi-dimensional Array

Once you have declared a multi-dimensional array, the next step is to actually create that array by allocating memory to it. The process to allocate memory to a multi-dimensional array is similar to allocating memory to a one-dimensional array except syntax. The syntax to create a multi-dimensional array is as follows:

```
var_name = new data_type [rows] [columns];
```

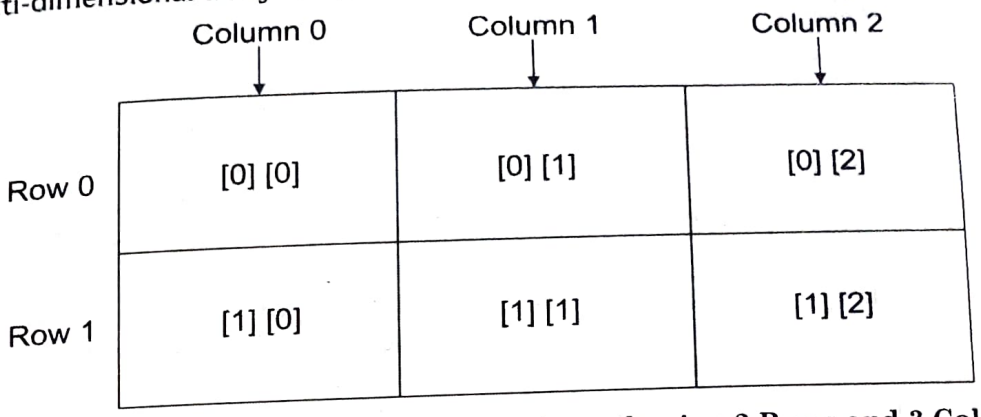
In the preceding syntax:

- **var_name** is the name of the multi-dimensional array declared in the preceding section.
- **new** is the operator, which is used to allocate memory to the array.
- **data_type** is the data type of the elements that will be stored in the array.
- **rows** are the total number of rows in a multi-dimensional array.
- **columns** are the total number of columns per row in a multi-dimensional array.

You can also combine the array's declaration and creation in a single step, as shown here:

```
int even_no [ ] [ ]= new int [2] [3];
```

Combining declaration and creation in a single step saves the time. The preceding example creates a multi-dimensional array of 2 rows and 3 columns, as shown in Figure 3:



▲ **Figure 3: Displaying a Multi-dimension Array having 2 Rows and 3 Columns**

Now, when a multi-dimensional array, even_no is created, it's time to initialize it.

Initializing an Array

The process to initialize a multi-dimensional array is similar to initializing a one-dimensional array. The syntax to initialize a multi-dimensional array is as follows:

```
var_name [ row_no ] [ col_no] = col_val;
```

In the preceding syntax:

- **var_name** is the array name.
- **row_no** is the row number in multi-dimensional array
- **col_no** is the column number in multi-dimensional array
- **col-val** is the value that will be stored at the specified column number of the specified row.

See the following example to initialize a multi-dimensional array:

```
even_no [0][0]= 2;
even_no [0][1]= 4;
even_no [0][2]= 6;
even_no [1][0]= 8;
even_no [1][1]= 10;
even_no [1][2]= 12;
even_no [2][0]= 14;
even_no [2][1]= 16;
even_no [2][2]= 18;
```

You can also initialize a multi-dimensional array at the time of declaration. If you initialize a multi-dimensional array at the time of declaration then you do not need to use the new keyword to assign memory to array elements. You also do not need to specify the size of an array. Syntax to initialize an array at the time of declaration is as follows:

```
data_type var_name [ ] [ ]= {{row0_col0, row0_col1, row0_coln}, { row1_col0,
 row1_col1, row1_coln}};
```

In the preceding syntax:

- **data_type** is the data type of a multi-dimensional array, which specifies the type of elements that will be stored in a multi-dimensional array.
- **var_name** is the name of the array.
- **row0_col0** in the first curly braces is representing an array element in 0^{th} row at 0^{th} column. One curly brace represents one row while the column number can be extended up to nth size.
- **row1_col0** in the second curly braces is representing an array element in 1^{th} row at 0^{th} column.

While initializing an array at the time of declaration, you also do not need to specify row numbers. This is because, the JVM at runtime itself calculates the row numbers based on the total number of the curly braces separated by commas in the multi-dimensional array. Therefore, the JVM internally assigns memory to the multi-dimensional array on the basis of the calculated array size. The following example initializes a multi-dimensional array, odd_no at the time of array declaration:

```
int odd_no [ ] [ ] = {{ 3,5,7}, {1, 7, 9}, {5,55, 35}};
```

A multi-dimensional array can also be called as even multi-dimensional array because the column numbers for all the rows remains same. In Java, you can also create a multi-dimensional array with varied column numbers.

Creating an Uneven Multi-dimensional Array

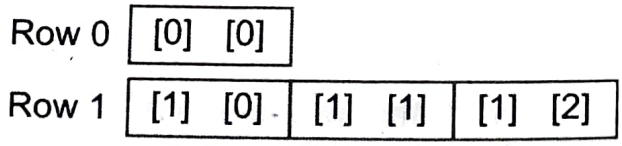
A multi-dimensional array with varied column numbers is known as uneven multi-dimensional array. The process to declare an uneven multi-dimensional array is same as of declaring an even multi-dimensional array. However, the process to create an uneven multi-dimensional array is different. See the following example to create an uneven multi-dimensional array is:

```
int muld [ ] [ ]=new int [2] [ ];
```

In the preceding example, you can see that only the row number is specified, so that you can specify column numbers manually, as shown in the following code snippet:

```
muld [ 0] = new int [1];
muld [ 1] = new int [3];
```

In the preceding code snippet, 1 column is assigned to the first row and 3 columns are assigned to the second row. Diagrammatic representation of the preceding code snippet is shown in Figure 4:



▲ **Figure 4: Uneven Multi-dimensional Array**

This is all about declaring, creating, and initializing a multi-dimensional array. Let's now discuss how a Java program uses a multi-dimensional array.

Developing a Program Using Multi-dimensional Array

In this section, we create a program using multi-dimensional array to display odd values stored in a multi-dimensional array. Listing 2 declares, creates, initializes, and uses a multi-dimensional array (you can find the ArrayExample2.java file on the CD in the code\chapter3 folder):

▶ **Listing 2: Program to Display Odd Values using a Multi-dimensional Array**

```
public class ArrayExample2
{
  public static void main(String args[])
  {
      int [][] muld= new int [2] [3];
      int i, j;
      int k=1;
      for (i=0;i<2;i++)
      {
          for (j=0;j<3;j++)
          {
              muld[i][j]=k;
              k=k+2;
          }
      }
      for(i=0;i<2;i++)
      {
          for(j=0;j<3;j++)
          {
              System.out.println(muld[i][j]+"");
          }
      }
  }
}
```

In Listing 2, a multi-dimensional array, `muld` of 2 rows and 3 columns are created. The multi-dimensional array is then initialized with first 6 odd values using for loops. The first 3 odd values are assigned in the first rows and remaining in the second row. The values are then displayed, as shown in Figure 5:

▲ **Figure 5: Displaying the Output of the ArrayExample2 Class**

This is all about declaring, creating, initializing, and using arrays of simple data types in Java. Besides creating arrays of simple data types, you can also create arrays of class objects in Java.

TEST YOUR KNOWLEDGE

Q16. Write a program to create3 x 4 matrix. Also display the sum and average of the matrix.

Ans.

```
class TestYourKnowledge16
{
 public static void main(String args[])
 {
     int A[][]={{1,2,3,4},{2,3,4,5},{3,4,5,6}};
     float sum=0.0f;
     int k=0;
     for (int I=0;I<=3;I++)
     for (int J=I+1;J<=3;J++)
     {
         sum+=A[I][J];
         ++k;
     }
     System.out.println("Sum of matrix is "+sum);
     System.out.println("Avg. of matrix is "+(float)sum/k);
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge16.java
D:\code\chapter3>java TestYourKnowledge16
```

Output:

```
Sum of matrix is 24.0
Avg. of matrix is 4.0
```

Q17. Write a program to implement GAUSS formula.

Ans.

```
class TestYourKnowledge17
{
 static float X[]=new float [10];
 static float A[][]=new float [10][10];
 static int N=3;
 static void input()
 {
     float B[][]={{0,0,0,0,0},{0,2,6,1,26},{0,3,1,2,17},{0,4,2,2,22}};
     for (int I=1;I<=3;I++)
     for (int J=1;J<=4;J++)
     A[I][J]=B[I][J];
}
static void pivot (int K)
{
    int L=K;
    for (int I=K+1;I<=N;I++)
```

```
        for ( int J=1;J<=N+1;J++)
        {
            float T=A[L][J];
            A[L][J]=A[K][J];
            A[K][J]=T;
        }
    }

    static void elimination()
    {
        for (int k=1;k<=N-1;k++)
        {
            pivot (k);
            for (int I=k+1;I<=N;I++)
            {
                float fact=(A[I][k]/A[k][k]);
                for (int J=1;J<=N+1;J++)
                A[I][J]=A[I][J]-A[k][J]*fact;
            }
        }
    }

        static  void backSub()

    {
        X[N]=A[N][N+1]/A[N][N];
        for (int I=N-1;I>=1;I--)
        {
            float SUM=0;
            for (int J=I+1;J<=N;J++)
            SUM+=A[I][J]*X[J];
            X[I] =A[I][N+1]-SUM/A[I][I];
        }
    }

    public static void main (String args[])
    {
        input();
        elimination();
        backSub();
        for (int I=1; I<=N;I++)
        System.out.println("  "+X[I]);
    }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge17.java
D:\code\chapter3>java TestYourKnowledge17
```

Output:

```
12.5
15.0
4.0
```

Q18. Write a program to multiply two one-dimensional arrays and put their values in a two-dimensional array.

Ans.

```
class TestYourKnowledge18
{
 public static void main(String args[])
 {
     int M=2;
     int L=4;
     int N=2;
     int A[]={0,1,2,3,4,5,6,7,8,9};
     int B[]={0,1,2,3,4,5,6,7,8,9};
     int C[][]=new int [3][3];
     int F,G,I,J,K;
     F=0;
     System.out.println("  ");
     for (I=1;I<=M;I++)
     {
         G=0;
         for (J=1;J<=N;J++)
         {
             C[I][J]=0;
             F=I*L-L;
             for (K=1;K<=L;K++)
             C[I][J] += A[++F]*B[++G];
         }
     }
     for (I=1;I<=N;I++)
     {

         System.out.println();
         for (J=1;J<=N;J++)
         System.out.print("  "+C[I][J]);
     }
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge18.java
D:\code\chapter3>java TestYourKnowledge18
```

Output:

```
30 70
70 174
```

Q19. Write a program to implement the concept of Interpolation.

Ans.

```
class INTERPOLATION
{
 static void TABLE (int N,float Y[],float T[][])
 {
     int I=0;
     for (I=0;I<=N;I++)
```

```
            T[I][0]=Y[I];
            for (int J=1;J<=N;J++)
            T[I][J]=T[I+1][J-1]-T[I][J-1];
        }
        float CALCULATE (int N,float X,float X0,float H,float T[][])
        {
            int I=0;
            float P=(X-X0)/H;
            float SUM=T[0][0];
            float TERM=1.0f;
            float F=1.0f;
            System.out.println(" " + TERM +" " + T[0][I]);
            SUM=SUM+TERM*T[0][I];
            F=F*(I+1);
            System.out.println("ROOT="+ SUM);
            return(SUM);
        }
    }
    class TestYourKnowledge19
    {
      public static void main (String args[])
      {
          int N=7;
          float H=1.0f,X0=1.0f,X=2.5f;
          float Y[]={3,6,9,12,15,18,21,24};
          float T[][]=new float [10][10];
          INTERPOLATION obj=new INTERPOLATION();
          obj.TABLE (N,Y,T);
          for (int I=0;I<=7;I++)
          {
              System.out.println(" ");
              for (int J=0;J<=N-I;J++)
              System.out.print(" " + T[I][J]);
          }
          System.out.println(" ");
          float ROOT=obj.CALCULATE (N,X,X0,H,T);
          System.out.println("ROOT="+ROOT);
      }
    }
```

NCS859

Execution:

```
D:\code\chapter3>javac TestYourKnowledge19.java
D:\code\chapter3>java TestYourKnowledge19
```

Output:

```
3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
6.0  0.0  0.0  0.0 0.0 0.0  0.0
9.0  0.0 0.0 0.0 0.0 0.0
12.0 0.0 0.0 0.0 0.0
15.0 0.0 0.0 0.0
18.0 0.0 0.0
21.0 0.0
24.0
1.0 3.0
ROOT=6.0
ROOT=6.0
```

Q20. Write a program to check whether a matrix is symmetrical or not.

Ans.

```
class TestYourKnowledge20
{
 static boolean sym(int n,int a[][])
 {
     boolean flag=true;
     for(int i=1;i<=n-1;i++)
     for(int j=i;j<=n-1;j++)
     if(a[i][j]!=a[j][i])
     flag=false;
     return(flag);
 }
 public static void main(String args[])
 {
     int n=3;
     int a[][]= { {1,2,3},{2,5,6},{3,6,9}};
     if(sym(n,a))
     System.out.println("Matrix is symmetrical in nature");
     else
     System.out.println("Matrix is nonsymmetrical in nature");
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge20.java
D:\code\chapter3>java TestYourKnowledge20
```

Output:

```
Matrix is symmetrical in nature
```

Q21. Write a program to create a 3 x 4 matrix. Also display the sum of the elements in each row and the total sum of the elements in the matrix.

Ans.

```
class TestYourKnowledge21
{
 public static void main(String args [])
 {
     int i,j,sum,Gsum=0;
     int A[][]={{10,15,17,39},{16,26,39,60},{28,32,180,40}};
     for(i=0; i<=2; i++)
     {
         sum=0;
         System.out.println(" ");
         for(j=0; j<=3; j++)
         {
             System.out.println(" "+A[i][j]);
             sum+=A[i][j];
         }
         System.out.println("  "+sum);
         Gsum+=sum;
     }
```

```
            System.out.println("    "+Gsum);
        }
    }
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge21.java
D:\code\chapter3>java TestYourKnowledge21
```

Output:

```
10
15
17
39
81
16
26
39
60
141
28
32
180
40
280
502
```

Q22. Write a program to add two rectangular matrix.

Ans.

```
class TestYourKnowledge22
{
 public static void main(String args [])
 {
     int I,J;
     int A[][]={{8,12,13},{17,21,36},{49,56,63},{72,81,89}};
     int B[][]={{10,17,14},{36,49,17},{11,18,24},{24,28,32}};
     int C[][]=new int [4] [3];
     for(I=0; I<=3; I++)
     for(J=0; J<=2; J++)
     C[I][J]=A[I][J]+B[I][J];
     for(I=0; I<=3;I++)
     {
         System.out.println("  ");
         for(J=0; J<=2; J++)
         System.out.print(" " +C[I][J]);
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge22.java
D:\code\chapter3>java TestYourKnowledge22
```

Output:

```
18 29 27
53 70 53
60 74 87
96 109 121
```

Q23. Write a program to multiply two matrices.

Ans.

```
class TestYourKnowledge23
{
 public static void main(String args [])
 {
     int A[][]={{8,2,3},{2,4,6},{2,8,7},{4,1,3}};
     int B[][]={{4,2,3},{8,7,1},{6,4,2}};
     int C[][]=new int [4][3];
     for(int i=0; i<4; i++)
     {
         for(int j=0; j<3; j++)
         {
             C[i][j]=0;
             for (int k=0; k<3;k++)
             C[i][j]+=A[i][k]*B[k][j];
         }
     }
     for(int i=0; i<=3; i++)
     {
         System.out.println( );
         for(int j=0; j<3; j++)
         {
             System.out.println(" "+C[i][j]);
         }
     }
 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge23.java
D:\code\chapter3>java TestYourKnowledge23
```

Output:

```
66
42
32

76
56
22

114
88
28
```

```
42
27
19
```

Using Array of Objects

In Java, an array of objects can also be created to store and process the objects. The process to declare and create an array of objects is similar to the process of declaring and creating an array of simple data types. When we create an array of objects then the data type gets replaced with the class name. The following is the syntax to declare and create an array of objects:

```
class_name arr_name [ ] = new class_name [size];
```

In the preceding syntax:

- **class_name** is the name of the class whose objects are used to create an array of objects.
- **arr_name** is the name of the array, which is to be created.
- **new** is the Java operator, which is used to allocate memory to an array.
- **size** is the number of objects that an array can hold.

Let's now suppose that we have a class, ArrayExample3. We have to create an array of objects of this class. To create an array of objects to ArrayExample3 class see the following example:

```
ArrayExample3 a_obj [ ] = new ArrayExample3 [2];
```

The `a_obj` array can hold 2 objects of `ArrayExample3` class. To use the `a_obj` array, we have to create objects of every array index. See the following code snippet to create objects of every index of a_obj array:

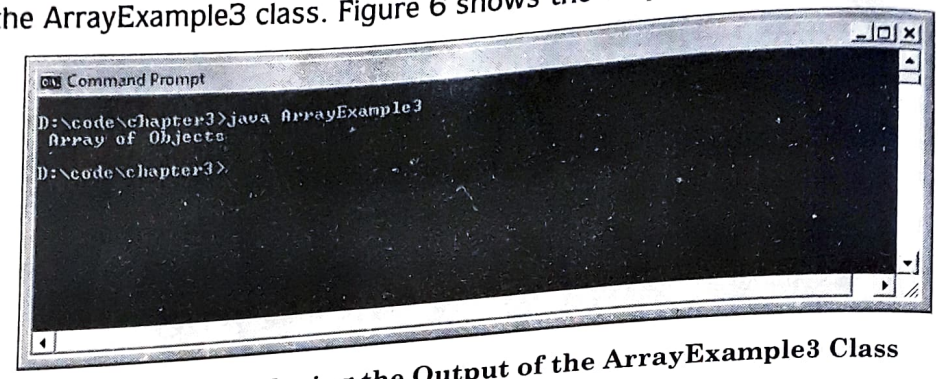
```
a_obj [ 0] = new ArrayExample3();
a_obj [ 1] = new ArrayExample3();
```

Listing 3 uses an array of objects (you can find the ArrayExample3.java file on the CD in the code\chapter3 folder):

▶ **Listing 3: Program to use an Array of Objects**

```
class ArrayExample3
{
 public void display()
 {
    System.out.println (" Array of Objects ");
 }
 public static void main (String args[])
 {
    ArrayExample3 a_obj[ ]=new ArrayExample3[3];
    a_obj[0]=new ArrayExample3();
    a_obj[1]=new ArrayExample3();
    a_obj[2]=new ArrayExample3();
    a_obj[1].display();
 }
}
```

In Listing 3, an array, a_obj is created to hold 3 objects of ArrayExample3 class. Then, object of every array index is created. After that the a_obj [1] index is used to invoke the display() method of the ArrayExample3 class. Figure 6 shows the output of Listing 3:



▲ **Figure 6: Displaying the Output of the ArrayExample3 Class**

In Java, you can also copy one array into another just by using a simple method rather than writing a ling program.

Using the System.arraycopy() Method

The arraycopy() is a method of the System class and used to copy an array elements from one array to another. The benefit of using this method is that you can copy the whole array into another array and array elements within a specified range as well. This method also saves your time that you spend in creating a long and complex program to copy an array into another array. The syntax to use System.arraycopy() method is:

```
System.arraycopy(Object first_array, int start_position, Object
 second_array, int start_pos2, int num_element);
```

In the preceding syntax:

- ***System.arraycopy*** is the method, which is used to copy one array into another array.
- ***Object first_array*** is the name of the first array.
- ***int start_position*** is the starting position in the first array from where the elements of the first array will be copied to second array.
- ***Object second_array*** is the name of the second array.
- ***int start_pos2*** is the starting position in the second array. It is the position from where the element of the first array starts copying. If any of the second array elements will be present at that location then, it will be overridden by elements of the first array.
- **int num_element** represents total number of first array elements that are to be copied into second array.

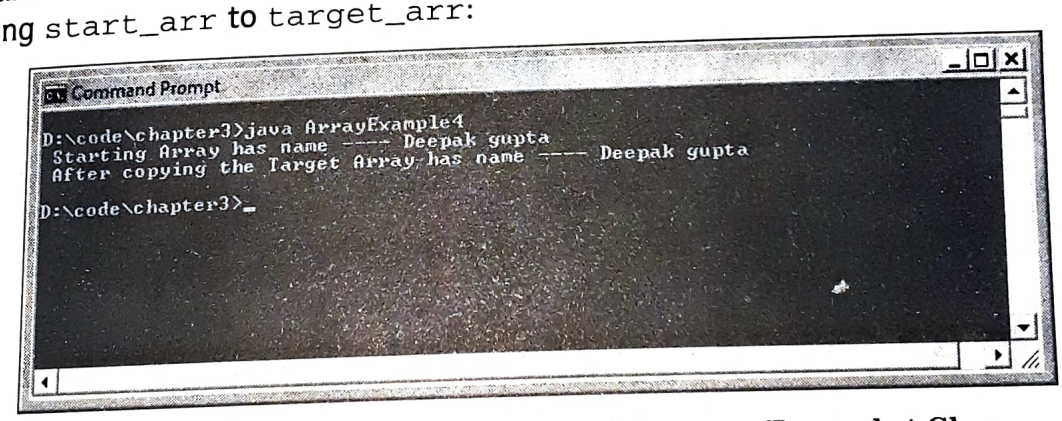
Listing 4 shows how to copy array elements into another array (you can find the ArrayExample4.java file on the CD in the code\chapter3 folder):

▶ **Listing 4: Using the System.arraycopy() Method**

```
class ArrayExample4
{
 public static void main (String args[])
 {
    char start_arr[ ] = { 'D','e','e','p','a','k',' ','g', 'u', 'p',
    't', 'a'};
```

```
        char target_arr[ ] = new char[15];
        System.out.print(" Starting Array has name ---- " );
        for( int x=0; x<start_arr.length; x++)
        {
            System.out.print(start_arr[x]);
        }
        System.out.print(" \n" );
        System.arraycopy(start_arr, 0, target_arr,0, 12);
        System.out.print(" After copying the Target Array has name ---- "
        );
        for( int y=0; y<target_arr.length; y++)
        {
            System.out.print(target_arr[y]);
        }
        System.out.print(" \n" );
    }
}
```

In Listing 4, two character type arrays named `start_arr` and `target_arr` are created. The array, `start_arr` is also initialized with some characters values while the array, `target_arr` is not initialized. The array, `target_arr` is not initialized because we copy the content of `start_arr` to `target_arr`. After creating both arrays, the value stored in `start_arr` is displayed. Then, the `System.arraycopy()` method is applied to copy `start_arr` to `target_arr`. Next, to check whether the `start_arr` is copied to `target_arr` or not, the content of `target_arr` is displayed. Figure 7 shows the result after copying `start_arr` to `target_arr`:



▲ **Figure 7: Displaying the Output of the ArrayExample4 Class**

This is all about creating and using arrays in Java. Till now, we have used fixed size arrays. Sometimes, fixed size arrays may lead to program failure because of the size of the value passed at runtime is more than the array size. To get rid of such problems, Java provides us a class, Vector to create dynamic arrays.

TEST YOUR KNOWLEDGE

Q24. Write a program to create a character array to store 6 characters. Also initialize the array with 6 random characters. Now, create another array containing 10 characters. Copy the elements ranging from index 2 to 4 of first array to second array at the same index.

Ans.

```
class TestYourKnowledge24
{
public static void main (String args[])
{
```

```
char first_arr[ ] = { 'K','D','E','M','M','k'};
char second_arr[ ] = {'B','C','R','H','N','M','F','D','X','G'};
System.out.print(" Elements in first array in between index 2 to 4 are
  --- " );
for( int x=2; x<=4; x++)
{
  System.out.print(first_arr[x]);
}
System.out.print(" \n" );
System.out.print(" Elements in second array in between index 2 to 4
  are --- " );
for( int j=2; j<=4; j++)
{
  System.out.print(second_arr[j]);
}
System.arraycopy(first_arr, 2, second_arr,2, 3);
System.out.print(" \n" );
System.out.print("Elements in second array after coying are ---- " );
for( int y=2; y<=4; y++)
{
  System.out.print(second_arr[y]);
}
System.out.print(" \n" );
}
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge24.java
D:\code\chapter3>java TestYourKnowledge24
```

Output:

```
Elements in first array in between index 2 to 4 are --- EMM
Elements in second array in between index 2 to 4 are --- RHN
Elements in second array after copying are ---- EMM
```

3.3 Using Vectors in Java

In Java, you can create an array whose size can be increased or decreased dynamically. Such kind of array is known as vector. The class used to create a dynamic array is called `Vector`. However, the elements of a dynamic array are accessed using the methods of `Enumeration` interface. In this section, we study about the `Enumeration` interface and `Vector` class under the following sub-headings:

- Defining the Enumeration Interface
- Defining the Vector Class
- Demonstrating Vector in a Program

Let's discuss these one by one in the following section.

Defining the Enumeration Interface

The Enumeration interface is an interface that allows you create a series of elements called enumeration. The `Enumeration` interface also helps you to iterate over the elements of classes such as, `Vector`, `Stack`. Table 1 explains the methods of the Enumeration interface:

Table 1: Methods of the Enumeration Interface

Method	Explanation
`boolean hasMoreElements()`	Returns true if an enumeration contains more elements to iterate else returns false
`Object nextElement()`	Gets the next element of an enumeration

Defining the Vector Class

The Vector class is defined in the java.util package. The Vector class implements a vector, a resizable array of objects. A vector always has some initial capacity to store elements. When this initial capacity is reached then the size of the vector automatically increases. The following are the advantages of using vector over arrays:

- A vector can contain heterogeneous objects
- You can add or remove objects from a vector according to the requirements

A limitation of using vectors is that you cannot directly store elements of primitive data types in a vector. A vector can only store objects; therefore, elements of primitive data types first need to be converted into objects. An element of primitive data type is converted into an object using the `Wrapper` classes that are discussed later in this chapter. Table 2 lists the fields of the Vector class:

Table 2: Fields of the Vector Class

Field	Explanation
`protected int capacityIncrement`	This is the amount by which the capacity of a vector increases when its size exceeds its capacity
`protected int elementCount`	Total number of components in this Vector object
`protected Object[] elementData`	The array where the components of the vector are stored

Table 3 lists the constructors of the Vector class.

Table 3: Constructors of the Vector Class

Constructor	Explanation
`Vector()`	Constructs an empty vector.
`Vector(Collection c)`	Constructs a vector containing the elements of the given collection.
`Vector(int initialCapacity)`	Constructs an empty vector with the given initial capacity.
`Vector(int initialCapacity, int capacityIncrement)`	Constructs an empty vector with the given initial capacity and capacity increment. When the initial capacity is reached then the vector is resized according to the specified increment capacity.

Table 4 lists the methods of the `Vector` class.

Table 4: Methods of the Vector Class

Method	Explanation
`void add(int index, Object element)`	Inserts the given element object at the specified location in the invoking vector.
`boolean add(Object element)`	Adds the given element object to the end of the invoking vector. It returns true if the element object is added successfully else returns false.
`boolean addAll (Collection c)`	Adds all the elements of the given collection to the end of the invoking vector. It returns true if the elements of the given collection are added successfully else returns false.
`boolean addAll(int index, Collection c)`	Adds all the elements of the given collection in the invoking vector at the given position. It returns true if the elements of the given collection are added successfully else returns false.
`void addElement(Object element)`	Adds the given element object to the end of the invoking vector.
`int capacity()`	Returns the current capacity of the invoking vector.
`void clear()`	Removes all the elements from the invoking vector.
`Object clone()`	Returns a clone of the invoking vector.
`boolean contains (Object element)`	Returns true if the invoking vector contains the object of the given element else returns false.
`boolean containsAll (Collection c)`	Returns true if the invoking vector contains all the elements of the given collection.
`void copyInto(Object[] anArray)`	Copies the objects of the invoking vector into the given array.
`Object elementAt(int index)`	Returns the element object located at the given index in the invoking vector.
`Enumeration elements()`	Returns an enumeration of the objects of the invoking vector.
`void ensureCapacity (int minCapacity)`	Ensures the minimum capacity of the invoking vector by increasing the vector size, if necessary.
`boolean equals(Object element)`	Compares the given element object with the invoking vector for equality. Returns true if the given element object is equal to the invoking vector object else returns false.
`Object firstElement()`	Returns the first element object of the invoking vector.
`Object get(int index)`	Returns the element object at the given position in the invoking vector.
`int hashCode()`	Returns the hashcode value of the invoking vector.

Table 4: Methods of the Vector Class

Method	Explanation
`int indexOf(Object element)`	Returns the index of the first occurrence of the given element in the invoking vector. Returns -1 if the invoking vector does not contain the given element.
`int indexOf (Object element, int index)`	Searches for the first occurrence of the given element object from the given index.
`void insertElementAt (Object element, int index)`	Inserts the given element object in the invoking vector at the given index.
`boolean isEmpty()`	Tests whether the invoking vector is empty or not. It returns true if the invoking vector is empty else returns false.
`Object lastElement()`	Returns the last element object of the invoking vector.
`int lastIndexOf (Object element)`	Returns the index of the last occurrence of the specified element object in the invoking vector. Returns -1 if the invoking vector does not contain the given element object.
`int lastIndexOf (Object element, int index)`	Searches in the backward direction for the given element object from the given index. It returns the last index of occurrence of the given element object. If the given element object is not found then it returns -1.
`Object remove(int index)`	Removes the element object at the given index in the invoking vector.
`boolean remove(Object element)`	Removes the first occurrence of the given element object. It returns true if the element is removed successfully else returns false.
`boolean removeAll (Collection c)`	Removes all the elements of the given collection from the invoking vector. It returns true if all the elements are removed successfully else returns false.
`void removeAllElements()`	Removes all element objects from the invoking vector and sets its size to zero
`boolean removeElement(Object element)`	Removes the first occurrence of the element object in the invoking vector. Returns true if the element object is removed else returns false.
`void removeElementAt (int index)`	Removes the element object at the given index in the invoking vector.
`protected void removeRange (int fromIndex, int toIndex)`	Removes all the element objects whose indexes are between fromIndex and toIndex from the invoking vector.

Table 4: Methods of the Vector Class

Method	Explanation
`boolean retainAll (Collection c)`	Keeps the elements in the invoking vector that are contained in the given collection.
`Object set (int index, Object element)`	Replaces the element at the given position in the invoking vector with the given element object.
`void setElementAt (Object obj, int index)`	Sets the given object at the given index in the invoking vector.
`void setSize(int newSize)`	Sets the size of the invoking vector.
`int size()`	Returns the total number of element objects contained by the invoking vector.
`List subList(int fromIndex, int toIndex)`	Returns a list of the element objects between fromIndex and toIndex in the invoking vector.
`Object[] toArray()`	Returns an array of element objects, which contains all the element objects of the invoking vector.
`String toString()`	Returns a string representation of the invoking vector.
`void trimToSize()`	Trims the capacity of the invoking vector to the total number of the element objects currently present in the invoking vector.

■ Demonstrating Vector in a Program

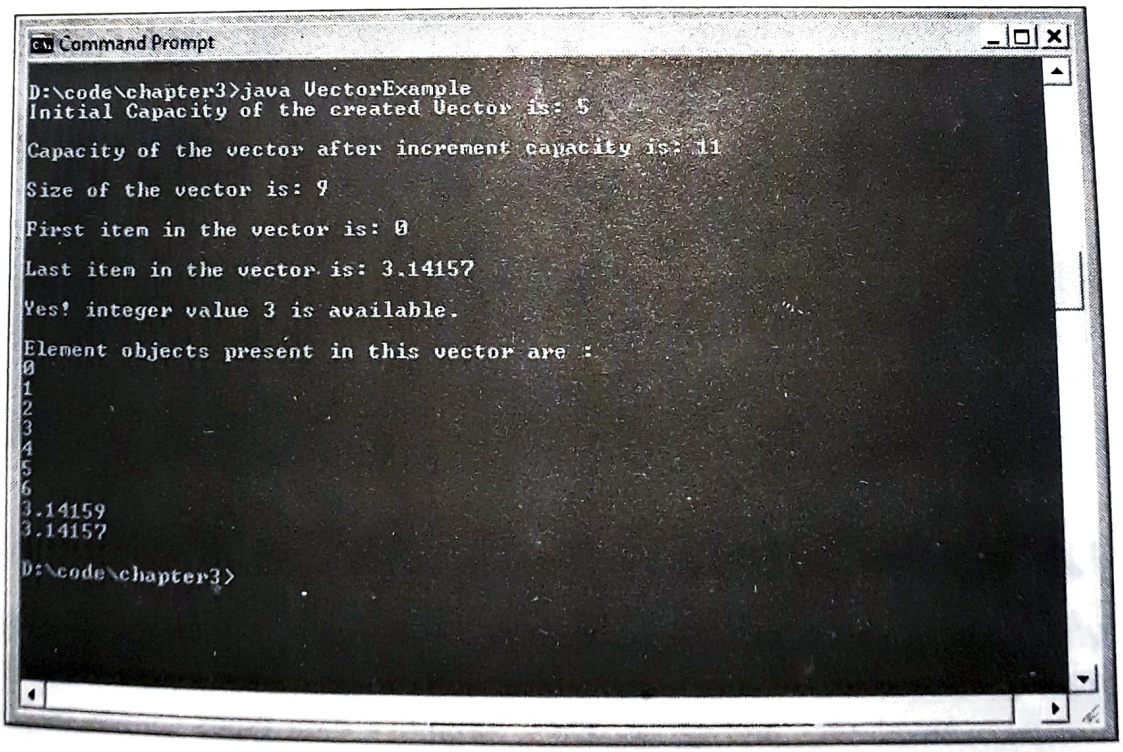
Each vector has an initial capacity, which specifies the size of a vector. When you exceed that size, the capacity of the vector automatically increases. You can use the `add()` and `addElement()` methods to add elements to a vector, the `remove()` and `removeElement()` methods to remove elements. You can use the `contains()` method for element search. Listing 5 declares the `Vector` class used to create a vector, check its size, capacity, and search for an element (you can find the VectorExample.java file on the CD in the code\chapter3 folder):

▶ ***Listing 5: Using the Vector Class***

```
import java.util.*;
class VectorExample {
 public static void main(String args[]) {
    Vector vect = new Vector(5,3);
System.out.println("Initial Capacity of the created Vector is: " +
 vect.capacity());
    vect.addElement(new Integer(0));
    vect.addElement(new Integer(1));
    vect.addElement(new Integer(2));
    vect.addElement(new Integer(3));
    vect.addElement(new Integer(4));
    vect.addElement(new Integer(5));
    vect.addElement(new Integer(6));
    vect.addElement(new Double(3.14159));
    vect.addElement(new Float(3.14157));
```

```
        System.out.println();
System.out.println("Capacity of the vector after increment capacity is: " +
  vect.capacity());
System.out.println();
System.out.println("Size of the vector is: " + vect.size());
  System.out.println();
System. out.println("First item in the vector is: " +
  (Integer)vect.firstElement());
System.out.println();
System.out.println("Last item in the vector is: " +
  (Float)vect.lastElement());
System.out.println();
if(vect.contains(new Integer(3)))
  {
      System.out.println("Yes! integer value 3 is available.");
  }
      System.out.println();
      // Traversing vector using Enumeration methods
      Enumeration e= vect.elements();
      System.out.println("Element objects present in this vector are : "  );
      while(e.hasMoreElements())
      {
              System.out.println(e.nextElement());
      }
  }
}
```

In Listing 5, a vector named `vect` of initial capacity 5 and increment capacity 3 is created. This vector is then initialized with heterogeneous element objects using the `addElement()` method. After that, vector's initial capacity, its size, first and last element in the vector, and presence of a particular element object in the vector are displayed. Next, an object, `e` of Enumeration interface is created, which is used with the methods of `Enumeration` interface to traverse and display the element objects stored in the vector, `vect`. Figure 8 displays the output of Listing 5:



▲ **Figure 8: Displaying the Output of the VectorExample Class**

With this, we have completed depicting the Vector class and its use. While discussing the Vector class, we told that the `Vector` class could only store element objects. Therefore, the elements of primitive data types must be converted into objects using their respective Wrapper class. Let's study converting primitive data types into objects.

TEST YOUR KNOWLEDGE

Q25. Write a program to create a dynamic array having initial capacity 5. Then, display the elements of that dynamic array and the size of the array.

Ans.

```
import java.util.*;
class TestYourKnowledge25
{
  public static void main(String args[])
  {
      Vector vect=new Vector(5,3);
      vect.addElement(new Integer(254));
      vect.addElement(new Double(214.25));
      vect.addElement(new Integer(142));
      vect.addElement(new Boolean(false));
      vect.addElement(new Integer(4));
      Enumeration e= vect.elements();
      System.out.println("Element objects present in this vector are :
          " );
      while(e.hasMoreElements())
      {
          System.out.println(e.nextElement());
      }
      System.out.println("Size of the dynamic array is: " +
          vect.size());
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge25.java
D:\code\chapter3>java TestYourKnowledge25
```

Output:

```
Element objects present in this vector are:
254
214.25
142
false
4
Size of the dynamic array is: 5.
```

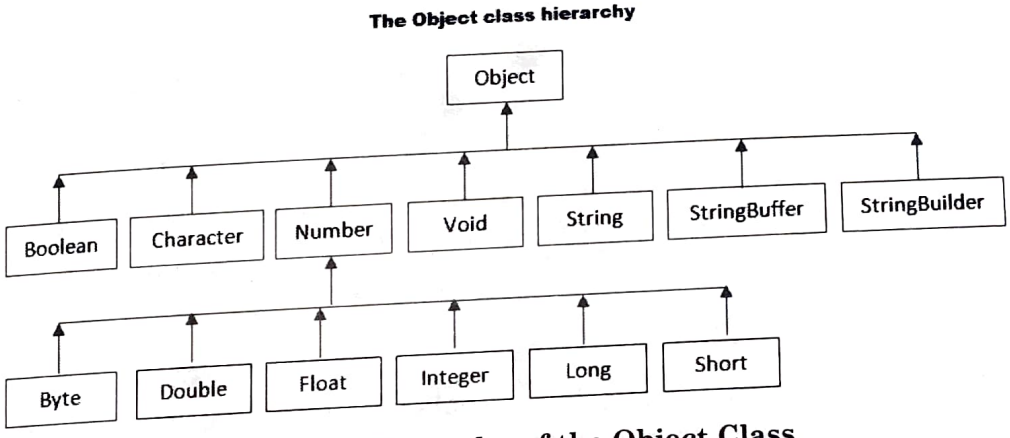
4 Using the Wrapper Classes in Java

Wrapper classes are the Java classes that convert the primitive data types into their respective objects. In Java, there are eight primitive data types: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, and `char`. These primitive data types cannot be treated as objects until you will not convert them into objects by using their respective `Wrapper` class. By converting

primitive data types into their respective objects. Wrapper classes serve the following two purposes:

- First, after conversion into objects, the primitive data types can be used in vectors.
- Second, the wrapper classes provide a variety of utility methods to convert any of the object into primitive types (such as byte, int, float, boolean, and char) or vice versa.

All wrapper classes belong to the java.lang package. The java.lang package is an indispensable part of Java programming. At compile time, the java.lang package is automatically imported to every source file. This package contains the Object class that is the parent of all classes. Boolean, Character and Number are child classes of the Object class, which are also wrapper classes. The Number class, is the parent for wrapper classes (Byte, Short, Integer, Long, Float, Double) to handle primitive values as objects. Figure 9 shows the class hierarchy of the Object class:



▲ **Figure 9: Hierarchy of the Object Class**

When we create an object of a wrapper class, it contains a field. In this field, we can store a primitive data type. It means, we can wrap or contain a primitive value into a wrapper class object. For example, if we create an object of Integer wrapper class, it contains a field int and it is possible to store an integer such as 34 or 89. Therefore, Integer is a wrapper class of int data type. Table 5 lists the primitive data types and their respective wrapper classes available in the java.lang package:

Table 5: Primitive Data Types and their Respective Wrapper Class

Primitive Data Types	Wrapper Classes	Constructor Arguments Taken by Wrapper Classes
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	Char
double	Double	double or String
float	Float	float, double or String
int	Integer	int or String
long	Long	long or String
short	Short	short or String

In this section, we depict the Wrapper classes and most commonly used methods of the Wrapper classes, under the following sub-headings:

- Using the Integer Wrapper Class
- Using the Double Wrapper Class
- Using the Character Wrapper Class
- Using the Boolean Wrapper Class
- Using Methods of the Wrapper Classes

Let's start studying Wrapper classes and their methods.

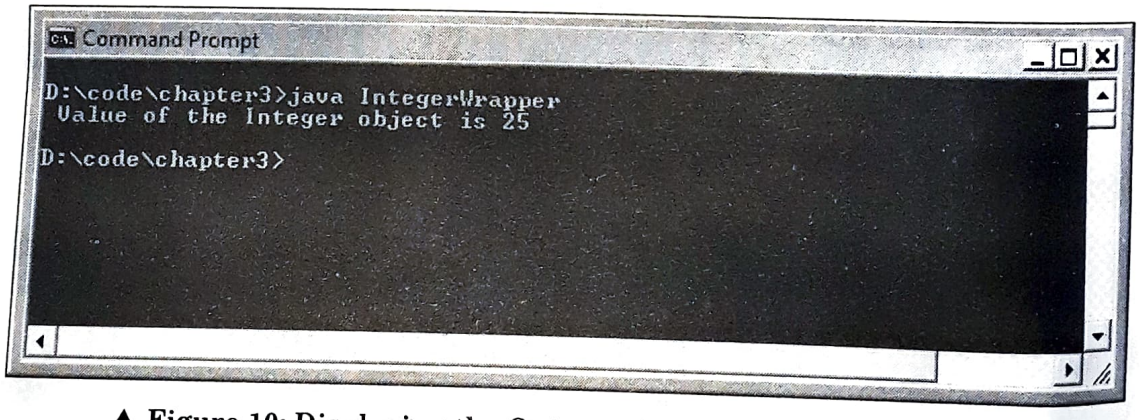
■ Using the Integer Wrapper Class

The `Integer` Wrapper class is the Java class that converts the value of primitive data type int into an object of the Integer Wrapper class. Listing 6 demonstrates the process of conversion of primitive data type int into an object of the Integer Wrapper class (you can find the IntegerWrapper.java file on the CD in the code\chapter3 folder):

▶ **Listing 6: Using the Integer Wrapper Class**

```
public class IntegerWrapper {
  public static void main(String args[]) {
      int n=25;
      Integer val = new Integer(n);
      System.out.println (" Value of the Integer object is " + val );
  }
}
```

In Listing 6, a variable `n` of primitive data type int is declared and initialized with the value 25. That integer value is then converted and stored into an object `val` of Integer Wrapper class. The value of the object val is displayed in Figure 10:



▲ **Figure 10: Displaying the Output of the IntegerWrapper Class**

■ Using the Double Wrapper Class

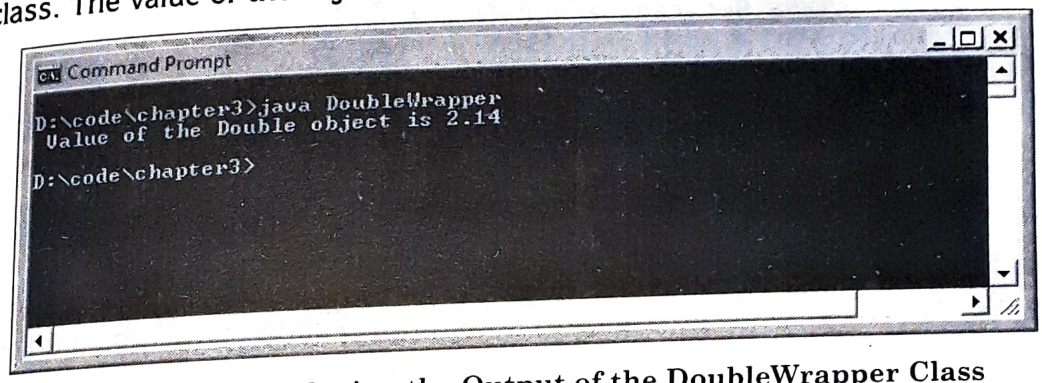
The `Double Wrapper` class is the Java class that converts the value of primitive data type double into an object of the Double Wrapper class. Listing 7 demonstrates the process of conversion of primitive data type double into an object of the Double Wrapper class (you can find the DoubleWrapper.java file on the CD in the code/chapter3 folder):

▶ **Listing 7: Using the Double Wrapper Class**

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

```
        double d=2.14;
        Double val = new Double(d);
        System.out.println (" Value of the Double object is " + val );
    }
}
```

In Listing 7, a variable `d` of primitive data type double is declared and initialized with the value 2.14. That double value is then converted and stored into an object `val` of the Double Wrapper class. The value of the object val is displayed in Figure 11:



▲ **Figure 11: Displaying the Output of the DoubleWrapper Class**

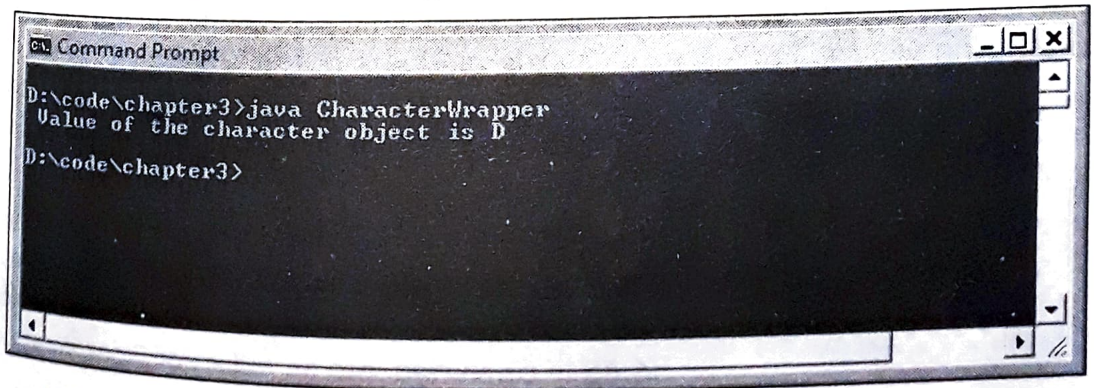
Using the Character Wrapper Class

The `Character Wrapper` class is the Java class that converts the value of primitive data type char into an object of the Character Wrapper class. Listing 8 demonstrates the process of conversion of primitive data type char into an object of the Character Wrapper class (you can find the CharacterWrapper.java file on the CD in the code\chapter3 folder):

▶ **Listing 8: Using the Character Wrapper Class**

```
public class CharacterWrapper {
 public static void main(String args[]) {
    char c='D';
    Character val = new Character(c);
    System.out.println (" Value of the character object is " + val );
 }
}
```

In Listing 8, a variable `c` of primitive data type char is declared and initialized with the value D. That character value is then converted and stored into an object `val` of Character Wrapper class. The value of the object val is displayed in Figure 12:



▲ **Figure 12: Displaying the Output of the CharacterWrapper Class**

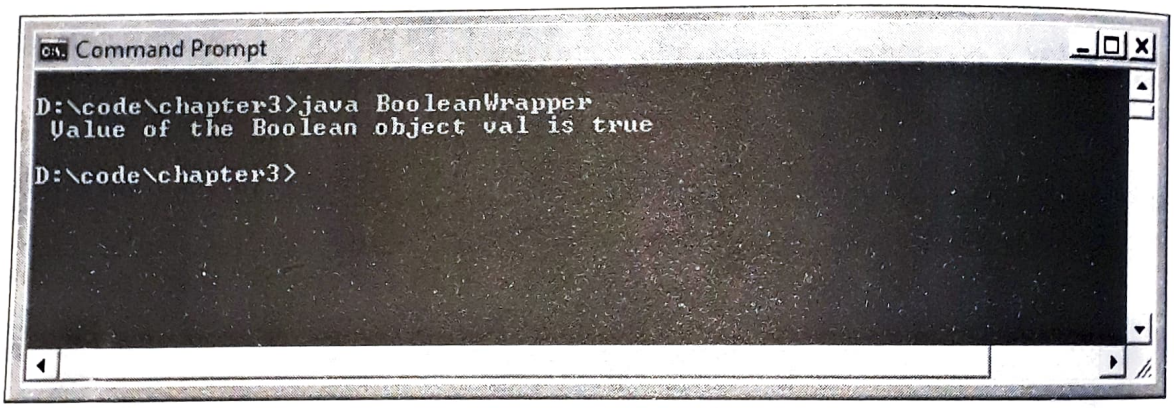
Using the Boolean Wrapper Class

The `Boolean Wrapper` class is the Java class that converts the value of primitive data type boolean into an object of the Boolean Wrapper class. Listing 9 demonstrates the process of conversion of primitive data type boolean into an object of the Boolean Wrapper class (you can find the BooleanWrapper.java file on the CD in the code\chapter3 folder):

▶ **Listing 9: Using the Boolean Wrapper Class**

```
public class BooleanWrapper {
  public static void main(String args[]) {
      boolean b=true;
      Boolean val = new Boolean(b);
      System.out.println (" Value of the Boolean object is " + val);
  }
}
```

In Listing 9, a variable `b` of primitive data type boolean is declared and initialized with the value, true. That boolean value is then converted and stored into an object `val` of the Boolean Wrapper class. The value of the object val is displayed in Figure 13:



▲ **Figure 13: Displaying the Output of the BooleanWrapper Class**

The value stored in the boolean primitive data type is case sensitive and must not be stored in " ". For example, boolean b=True and boolean b="true" generate compilation error.

Using Methods of the Wrapper Classes

The Wrapper classes in Java contain methods such as, `valueOf()`, `toString()`. These methods are used for different purposes, for example, the valueOf() method is used to convert a string value into an object. While the toString() method is used to convert Wrapper objects into String values. Table 6 explains the methods of the Wrapper Classes:

Table 6: Methods of the Wrapper Classes

Method	Explanation
`valueOf(String arg)`	Takes a single argument of String type. The argument is then converted into a Wrapper object. For example, `Float f = Float.valueOf("34.45");` In this example, the String value 34.45 is converted into an object of Float Wrapper class. This is a static method.

Table 6: Methods of the Wrapper Classes

Method	Explanation
`valueOf(String arg1, int arg2)`	Takes two arguments: one argument is of String type, while the other argument is of int type, which can be 2, 8, or 16. The number 2 represents binary value, 8 represents octal value, and 16 represents hexadecimal value. This method converts the String object into a Wrapper object based on the 2nd argument. For example, if the second argument is 2 then the binary equivalent of the String value is created.
`toString()`	Converts the Wrapper objects into String values. For example, `Float f = Float.valueOf("34.45");` `f.toString();` Where, it converts the object f into a String value.
`parseXxx()`	Accepts a String value and converts that value into primitive data type. It is also called as the parser method in which Xxx represents a Wrapper class. For example, `parseInt()`, `parseDouble()`, and `parseFloat()`. For example, `Double d = parseDouble("34.45")`, converts the String value 34.45 into primitive data type double.
`xxxValue()`	Converts a numeric Wrapper object into any primitive data type. This method can be used as `byteValue()`, `intValue()`, `doubleValue()`, `floatValue()`, `shortValue()`, and `longValue()`. All these methods do not take any argument. For example, `Integer val = new Integer(25);` `byte b= val.byteValue();` // converts the value stored in object val into bytes.
`toXxxString()`	Converts a numeric value into a String object. The converted value can be binary, octal, or hexadecimal equivalent of the numeric value. This method can be `toBinaryString()`, `toHexString()`, or `toOctalString()`. For example, `String s1 = Integer.toBinaryString(65)`, converts the value 65 into a String object.

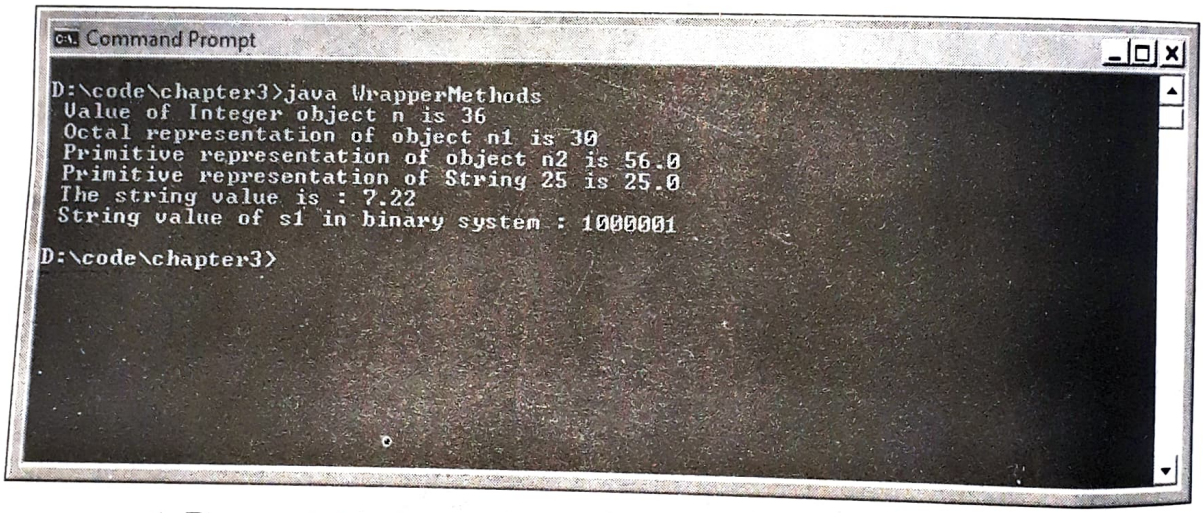
These are the methods of the Wrapper classes. Listing 10 shows the implementation of these methods (you can find the WrapperMethods.java file on the CD in the code\chapter3 folder):

▶ **Listing 10: Using Methods of the Wrapper Classes**

```
public class WrapperMethods {
  public static void main(String str[]) {
     // Using the valueOf() method with single argument
     Integer n = Integer.valueOf("36");
     System.out.println (" Value of Integer object n is " + n);
     // Using the valueOf() method with two arguments
     Integer n1 = Integer.valueOf("36",8);
     System.out.println (" Octal representation of object n1 is " + n1);

     // Using the XxxValue() method
     Integer n2= new Integer(56);
```

```
        double d= n2.doubleValue();
        System.out.println (" Primitive representation of object n2 is " +
        d);

        // Using the parseXxx() method
        float f= Float.parseFloat("25");
        System.out.println (" Primitive representation of String 25 is " +
        f);

        // Using the toString() method
        String d1 = Double.toString(7.22);
        System.out.println(" The string value is : "+ d1);
        // Using the toXxxString() method
        String s1 = Integer.toBinaryString(65);
        System.out.println(" String value of s1 in binary system : "+s1);
    }
}
```

Listing 10 is using all the methods listed in Table 6. First of all the `valueOf()` method is used to convert a numeric string value into an integer object. Then, the valueOf() method with two arguments are used to convert a numeric string value into its octal representation. Next, the `doubleValue()` method is used to convert an Integer object into a double primitive type, which is then followed by `parseFloat()` method to convert a numeric string into float data type. After that, the `toString()` method is used to convert a double value into a string value. In the last, the `toBinaryString()` method is used to get the binary equivalent of a numeric value. Figure 14 shows the output of Listing 10:



▲ **Figure 14: Displaying the output of the WrapperMethods Class**

Here, we have completed studying Wrapper classes in Java. In this section, you must have noticed that we have used a word string at so many places. Strings in Java are very important as they help you work with long textual information. Let's now study the String class, next.

.5 Using Strings in Java

Strings are very important in the world of programming languages as they are used to process long textual/symbolic information. For example, when you visit a bank to open an account then you are asked for information such as your name, address. The information you provide is in

the form of ordered sequence of characters and symbols. This ordered sequence of characters and symbols is known as `string`.

In almost all programming languages such as, C, C++, strings are considered as fundamental data types. In languages such as, C, C++, character arrays are used to process long textual information. Using character arrays to process long textual information is a bit tedious and time consuming. Therefore, in Java, strings are not considered as the fundamental data types. In Java, the String class is provided to work with strings along with the facility of character arrays. In Java, you can either use character arrays or the String class as per your requirement.

In Java, strings are immutable as once an object of the String class is created and initialized with some value then it cannot be changed. However, you may sometimes need to manipulate the value assigned to a string object. To resolve the problem of manipulating string, the StringBuffer class is provided in Java. In this section, we study the String class, methods of the String class, and the StringBuffer class under the following sub-headings:

- Defining the String Class
- Defining Arrays of the String type
- Implementing Strings Concatenation
- Using Methods of the String Class
- Comparing the equals() Method and == Operator
- Using the StringBuffer Class

Let's start studying strings in Java.

Defining the String Class

In Java, we have `int` to store integers, `float` to store floating- point numbers and so on, but there is no primitive type to store the string data. To handle string data in Java, we require objects of the String class. The String class is included in the java.lang package. Therefore, each class in Java can use the String class without importing any package. Table 7 lists the most commonly used constructors of the String class to let you work with String objects as per your need:

Table 7: Constructors of the String Class

Constructor	Explanation
`String ()`	Constructs a String object with no value. For example, `String str= new String();`
`String( String str)`	Constructs and initializes a String object with the given string value. For example, String `str=new String("Hello");` Now, the object `str` contains Hello.
`String ( char charArr [ ])`	Constructs and initializes a String object with the values of given character array. For example, `char item [ ] = {'C', 'O', 'P', 'Y'};` `String testString = new String (item);` Now, the object testString contains the value stored in the character array name item.

Table 7: Constructors of the String Class

Constructor	Explanation
`String (Char charArr[ ], int startIndex, int count)`	Constructs and initializes a String object with a part of a character array. For example, String `inputArray [ ] = "'P','U','S','H','P', 'O','P'";` `String testArr = new String(inputArray, 4, 3);` Now, the object `testArr` contains the value `POP` because the starting index is `4` and the total number of characters to be copied is `3`.
`String (byte byteArr[ ])`	Constructs and initializes a String object with an array having `ASCII` values. Apart from constructing and initializing a String object, this constructor also decodes the given array values according to the system's default character set.
`String (byte byteArr[ ], int startIndex, int count)`	Constructs and initializes a String object with a part of an array having `ASCII` values. This constructor also decodes the ASCII values according to the system's default character set.

Apart from using a constructor to create and initialize a String object, you can use the following code snippet to directly create and initialize a String object:

```
String str = "Hello";
```

The preceding code snippet creates a String object `str` and initializes it with the value `Hello`. You can also assign the value of a String object into another String object. See the following code snippet to assign value of a String object to another String object:

```
String str1="How";
String str2="Are";
str1=str2; // now the str1 contains the value Are
```

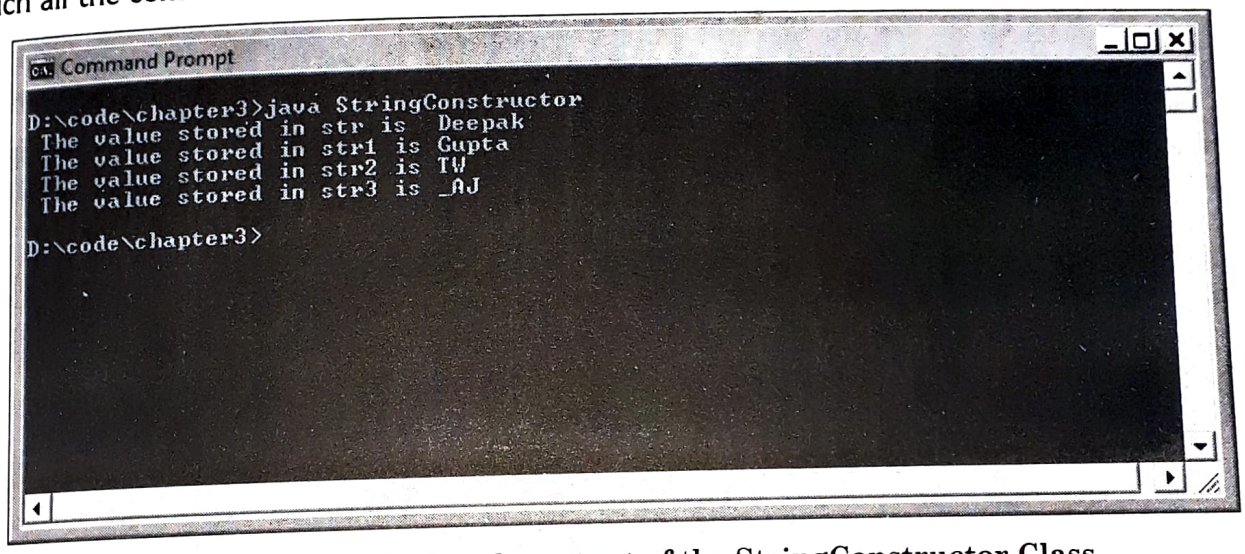
Listing 11 implements few of the `String` class constructors listed in Table 7 (you can find the StringConstructor.java file on the CD in the code\chapter3 folder):

▶ **Listing 11: Implementing Constructors of the String Class**

```
class StringConstructor {
  public static void main(String args[]) {
      String str= new String();
      str=" Deepak";
      System.out.println(" The value stored in str is " + str);
      String str1= new String ("Gupta");
      System.out.println(" The value stored in str1 is " + str1);
      char ch []= {'T','W'};
      String str2 = new String(ch);
      System.out.println(" The value stored in str2 is " + str2);
      byte byteArr[]={95,65,74};
      String str3=new String (byteArr);
      System.out.println(" The value stored in str3 is " + str3);
```

```
    }
}
```

In Listing 11, four constructors are used to create and initialize `str`, `str1`, `str2`, and `str3` String objects. The first constructor creates an empty String object `str`, which is then manually initialized with the value `Deepak`. The second constructor creates `str1` and initializes it with the value `Gupta`. The third constructor creates `str2` and initializes it with the value stored in the character array, `ch`. The fourth constructor creates `str3` and initializes it with the `ASCII` values stored in `byteArray`, `byteArr`. Figure 15 shows the values with which all the constructors are initialized:



▲ **Figure 15: Displaying the output of the StringConstructor Class**

Defining Arrays of the String Types

In Java, apart from creating a single String object, you can also create an array containing multiple strings. See the following code snippet, which defines and initializes an array named teamArray:

```
String teamArray = new String [5]; // declaring and creating String array
// initializing the String array
 teamArray[0] = "Niraj";
 teamArray[1] = "Umar";
 teamArray[2] = "Mahtab";
 teamArray[3] = "Shivam";
 teamArray[4] = "Vikash";
```

In the preceding code snippet, an array named `teamArray`, which can hold 5 string values is created and initialized. In this code snippet, each array index is initialized individually, which is a time consuming and tedious process. To save your time, you can use the following code snippet to initialize an array at the time of array creation:

```
String [] teamArray = {"Niraj", "Umar", "Mahtab", "Shivam", "Vikash"};
```

Listing 12 shows the implementation of a String array (you can find the StrArray.java file on the CD in the code/chapter3 folder):

▶ **Listing 12: Using a String Array**

```
class StrArray
{
  public static void main(String args[])
  {
      String [] teamArray = {"Niraj", "Umar", "Mahtab", "Shivam", "Vikash"};
      for(int i = 0; i<teamArray.length;i++)
      {
      System.out.print("Value at Index " + "[" +   i + "]"+ " is" +" : " );
          System.out.println(teamArray[i]);
      }
  }
}
```

In Listing 12, a String array named `teamArray` is created and initialized with 5 string values. The values stored in the array are then accessed using the for loop. Figure 16 shows the values stored in the array, `teamArray`:

▲ **Figure 16: Displaying the Output of the StrArray Class**

TEST YOUR KNOWLEDGE

Q26. Write a program to create an array of company name and the price quoted. Also fetch the company name who has quoted the lowest amount.

Ans.

```
///CREATE ARRAY OF COMPANY NAME & LOWEST PRICE QUOTED
class TestYourKnowledge26
{
 public static void main(String args [])
 {
     int N,I;
     String comp[]={"KMC","NMC","SMC","MCGM","PMC"};
     int price[]={45,69,58,39,98};
     N=price.length;
     int small=price[0];
     for (I=1;I<=N-1;I++)
     if (small>price[I])
     small=price[I];
     for (I=0;I<=N-1;I++)
     if (small==price[I])
```

▲ Figure 17: Displaying the Output of the StringConcat Class

➡ Strings Concatenation with Other Data Types

In Java, string values can be easily concatenated with the values of other data types using the concatenation operator, +. Listing 14 demonstrates the use of concatenation operator (you can find the ConcatString.java file on the CD in the code\chapter3 folder):

▶ **Listing 14: Concatenating String Values with Other Data Types**

```
public class ConcatString {
  public static void main(String args[]) {
      int date=25;
      String month = "July";

      int year= 1983;
      System.out.println(date + " " + month + " " + year);
  }
}
```

In Listing 14, two variables, `date` and `year` of data type `int` are created along with a String object, `month`. The variable date is initialized with the value `25` while the variable year is initialized with the value `1983`. The String object month is initialized with the value `July`. Figure 18 displays these values after concatenation:

▲ Figure 18: Displaying the Output of the ConcatString Class

```
            System.out.println(comp[I]);
        }
    }
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge26.java
D:\code\chapter3>java TestYourKnowledge26
```

Output:

```
MCGM
```

Implementing Strings Concatenation

In Java, a string can be concatenated with other strings or with the elements of other data types such as integer or float. You may need to concatenate two or more strings or a string with a number or symbol to display some information as per requirement. For example, when you fill any online registration form then you provide your first name and second name in two separate columns. However, when you finish the form then you see your complete name as a single string. In this section, we see the processes to concatenate two strings and a string with elements of other data type under the following sub-headings:

- Concatenation of Strings
- String Concatenation with other Data types

Let's discuss these processes one by one in the following section.

Concatenation of Strings

Two or more strings in Java are concatenated using the concatenation operator, +. See the following code snippet in which two strings "Hello" and "World" are concatenated to form a single string HelloWorld:

```
String str ="Hello" + "World";
```

Listing 13 demonstrates the strings concatenation (you can find the StringConcat.java file on the CD in the code\chapter3 folder):

▶ Listing 13: Using the Concatenation Operator, +

```
public class StringConcat
{
  public static void main(String args[])
      {
      String FName= "Deepak";
      String SName = "Gupta";
      System.out.println (FName + " " + SName);
  }
}
```

In Listing 13, two String objects, `FName` and `SName` are initialized with the values `Deepak` and `Gupta`. These String objects are then concatenated using the concatenation operator. Figure 17 shows the concatenated strings:

Using Methods of the String Class

In Java, the String class provides various methods that enable you to convert the string into a character array, convert numbers into strings, search strings, create substrings, change the case of the string, get a string's length, compare strings, and much more. Some of the most commonly used methods of the String class that we study in this section are:

- Implementing the length() Method
- Implementing the toLowerCase() Method
- Implementing the toUpperCase() Method
- Implementing the substring() Method
- Implementing the concat() Method
- Implementing the replace() Method
- Implementing the trim() Method
- Implementing the charAt() Method
- Implementing the getChars() Method
- Implementing the toCharArray() Method
- Implementing the equals() and equalsIgnoreCase() Methods
- Implementing the regionMatches() Method
- Implementing the startsWith() and endsWith() Methods
- Implementing the compareTo() and compareToIgnoreCase() Methods
- Implementing the indexOf() and lastIndexOf() Methods
- Implementing the toString() Method
- Implementing the valueOf() Method

Let's start implementing the methods of the String class.

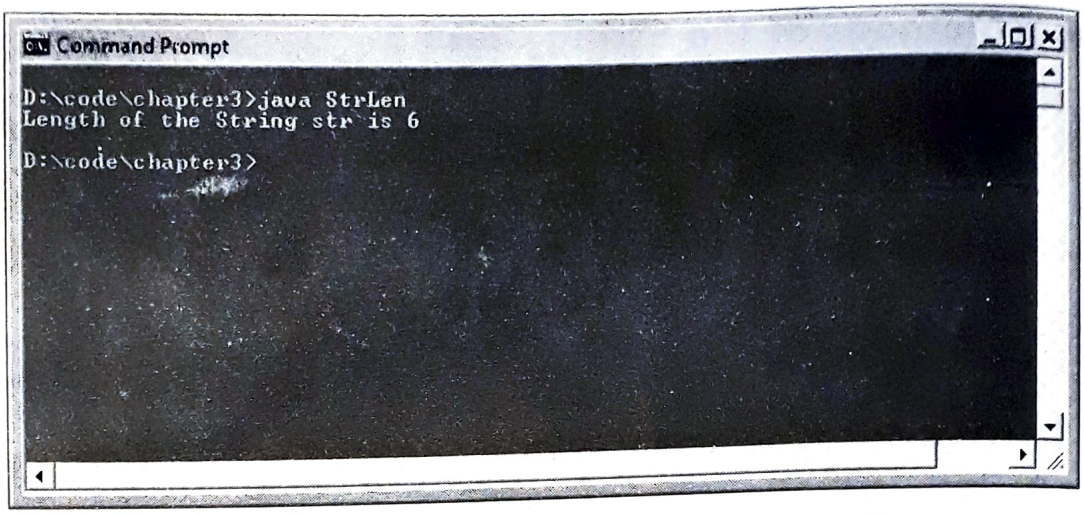
Implementing the length() Method

The `length()` method returns the length of a string. This method calculates all the characters, symbols, and numbers enclosed in " ". This method returns an integer value. Listing 15 demonstrates the implementation of the length() method (you can find the StrLen.java file on the CD in the code\chapter3 folder):

▶ **Listing 15: Using the length() Method**

```
public class StrLen{
  public static void main(String args[])
  {
      String str="length";
      int x= str.length();
      System.out.println("Length of the String str is " + x);
  }
}
```

In Listing 15, a String object `str` is initialized with the value, `length`. The length() method is then applied on `str` to calculate the length of str. Figure 19 displays the length of str:



▲ Figure 19: Displaying the Output of the StrLen Class

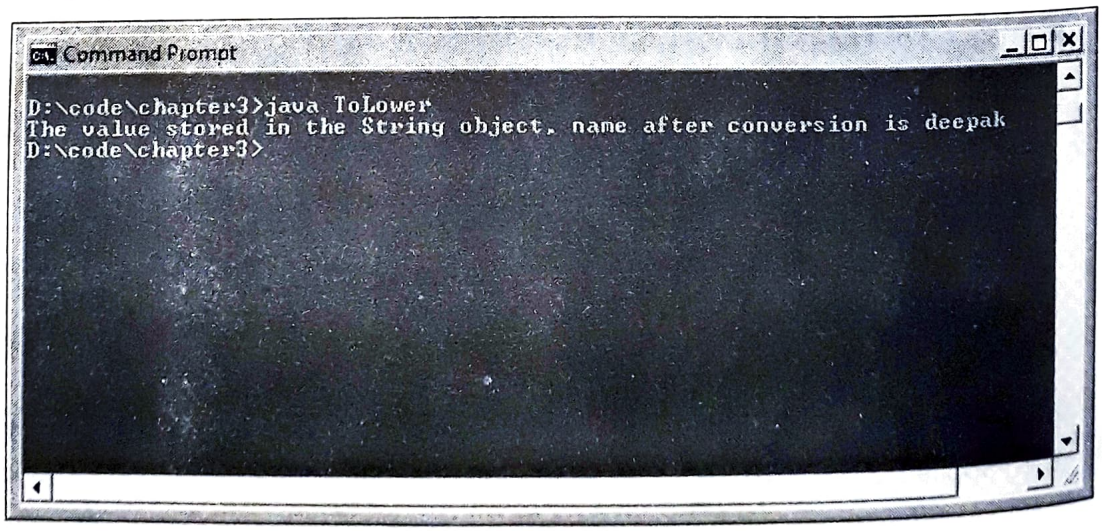
Implementing the toLowerCase() Method

The `toLowerCase()` method returns a string, after converting all the uppercase letters of the invoking string to lowercase. Listing 16 demonstrates the implementation of the toLowerCase() method (you can find the ToLower.java file on the CD in the code/chapter3 folder):

▶ **Listing 16: Using the toLowerCase() Method**

```
public class ToLower
{
  public static void main(String args[])
  {
      String name = "DEEPAK";
      System.out.print("The value stored in the String object, name after
      conversion is " + name.toLowerCase());
}
}
```

In Listing 16, a String object `name` is created and initialized with the value, `DEEPAK`. The toLowerCase() method is then applied on name to convert the value stored in it to lower case. Figure 20 displays the result after conversion:



▲ Figure 20: Displaying the Output of the ToLower Class

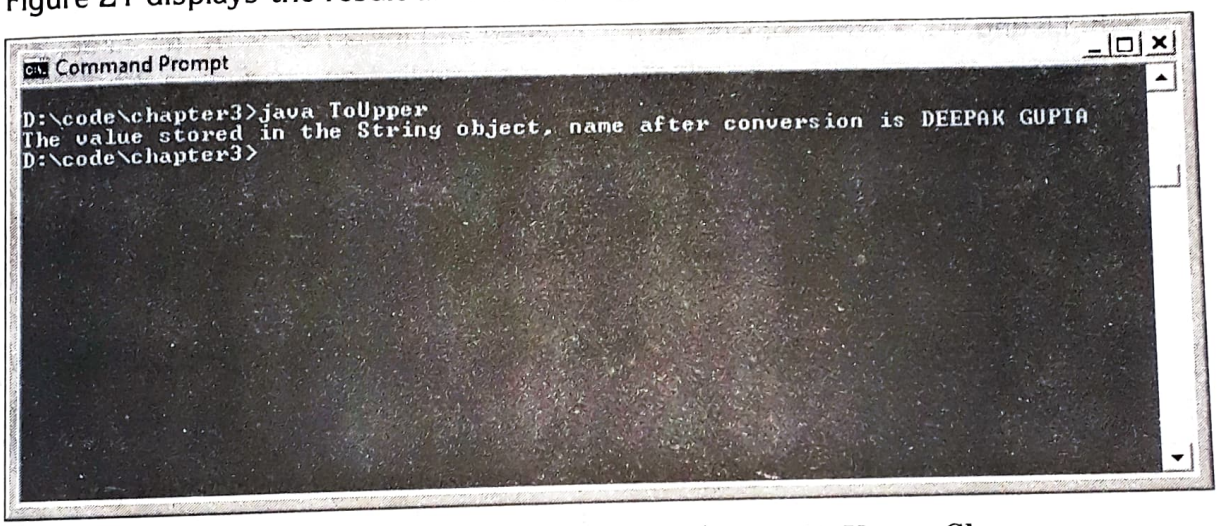
Implementing the toUpperCase() Method

The `toUpperCase()` method returns a string, after converting all the lowercase letters of the invoking string to uppercase. Listing 17 demonstrates the implementation of the toUpperCase() method (you can find the ToUpper.java file on the CD in the code\chapter3 folder):

▶ Listing 17: Using the toUpperCase() Method


```
public class ToUpper{
  public static void main(String args[]) {
  String name = "deepak gupta";
  System.out.print("The value stored in the String object, name after
      conversion is " + name.toUpperCase());
  }
}
```

In Listing 17, a String object, `name` is created and initialized with the value, `deepak gupta`. The toUpperCase() method is then applied on name to convert the value stored in it to upper case. Figure 21 displays the result after conversion:



▲ Figure 21: Displaying the Output of the ToUpper Class

Implementing the substring() Method

The `substring()` method returns a new String instance that is a substring of the original String instance. The `substring()` method is available in the following two forms:

- substring(int begin)
- substring(int begin, int end)

One form of the substring() function takes a single parameter, which creates a new String that begins at the index specified by the argument value and ends at the last character of the string. The following code snippet shows the use of the substring(int begin) method:

```
String str = "EDUCATION";
System.out.println (str.substring(4));
```

The preceding code snippet prints the value `ATION`. The second form of substring() method takes two parameters. It creates a substring that starts at the index of the first parameter and ends at the index of the second parameter. The character at the start index is included in the

substring, but the character at the end index is not. The following code snippet shows the use of the substring (int begin, int end) method:

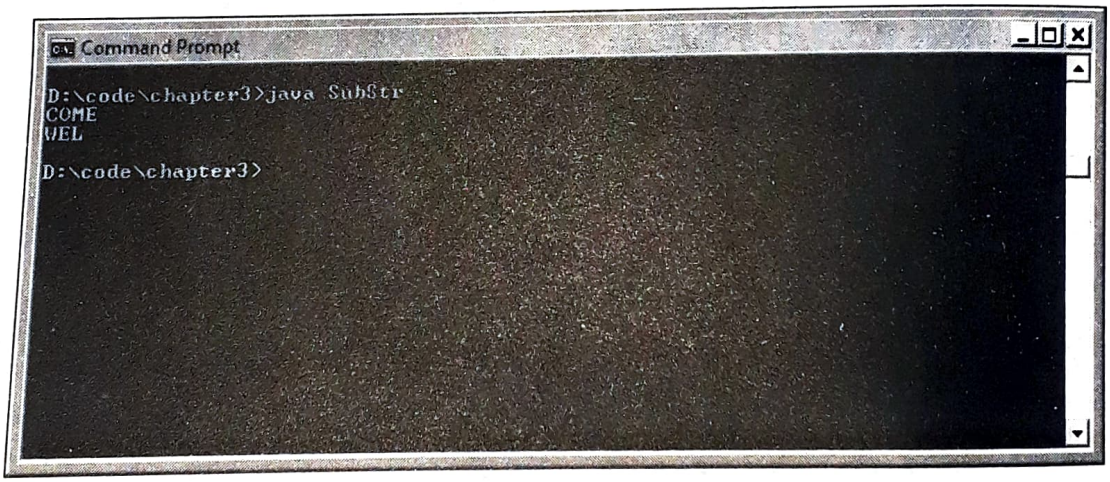
```
String str = "EDUCATION";
System.out.println (str.substring(3,6));
```

The preceding code snippet prints the value CAT. Listing 18 shows the implementation of both the forms of substring() method (you can find the SubStr.java file on the CD in the code\chapter3 folder):

▶ **Listing 18: Using the substring() Method**

```
public class SubStr{
  public static void main(String args[]) {
      String str = "WELCOME";
      System.out.println(str.substring(3));
      System.out.println(str.substring(0,3));
  }
}
```

In Listing 18, a String object, `str` is created and initialized with the value, WELCOME. Both the forms of the `substring()` method is then applied on str. Figure 22 displays the result after using the substring() method:



▲ Figure 22: Displaying the Output of the SubStr Class

Implementing the concat() Method

The `concat()` method appends a string at the end of another string. This method returns a string after concatenation. In Java, we can concatenate strings in two ways: first, using the concatenation operator, +; second, using the concat() method. The use of the concatenation operator has already been discussed in this chapter. Listing 19 demonstrates the implementation of the concat() method (you can find the Concat.java file on the CD in the code\chapter3 folder):

▶ **Listing 19: Using the concat()** `Method`

```
public class Concat{
  public static void main(String args[]) {
      String str = "Hello";
      String str1= "Java";
```

```
        System.out.print(str.concat(str1));
    }
}
```

In Listing 19, two String objects, `str` and `str1` are created and initialized with the values `Hello` and `Java`. These String objects are then concatenated using the concat(). Figure 23 shows the concatenated strings:

▲ **Figure 23: Displaying the Output of the Concat Class**

Implementing the replace() Method

In Java, the `replace()` method can be used for replacing a specific character in the invoking string with the given character and replacing a sub-string with the given sub-string. The replace() method can be used in three forms. Table 8 lists all the forms of the replace() method:

Table 8: Forms of the replace() method

Method	Explanation
`String replace(char oldChar, char newChar)`	Replaces all the occurrence of the old character with the new character
`String replaceAll(String oldString, String newString)`	Replaces all the occurrence of the old sub-string with the new sub-string
`String replaceFirst(String oldString, String newString)`	Replaces the first occurrence of the old sub-string with the new sub-string

Listing 20 shows the implementation of all the forms of the replace() method (you can find the Replace.java file on the CD in the code/chapter3 folder):

▶ **Listing 20: Using the replace() Method**

```
public class Replace{
  public static void main(String args[]) {
      String str = "Hello Java Hello";
      System.out.println(str.replace('H', ' ')); // first form
      System.out.println(str.replaceAll("Hello", "Hi")); // second form
      System.out.println(str.replaceFirst("Hello", "Hi")); // third form
  }
}
```

In Listing 20, a String object, `str` is created and initialized with the value `Hello Java Hello`. All the forms of the `replace()` methods are then applied on `str`. The first form replaces all the occurrence of the character `'H'` with a white space. The second form replaces all the occurrence of the sub-string `Hello` with `Hi`. While the third form replaces the first occurrence of the sub-string `Hello` with `Hi`. Figure 24 shows the output:

▲ **Figure 24: Displaying the Output of the Replace Class**

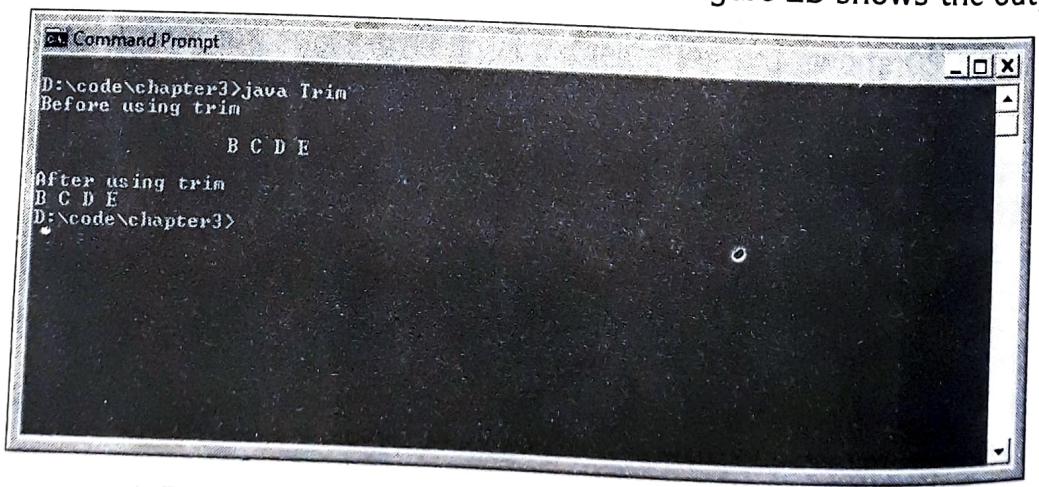
Implementing the trim() Method

The `trim()` method creates a new String instance with the leading and trailing white space removed. Listing 21 demonstrates implementation of the trim() method (you can find the Trim.java file on the CD in the code/chapter3 folder):

▶ **Listing 21: Using the trim() Method**

```
public class Trim {
  public static void main(String st[]) {
      String str = "     B C D E ";
      System.out.println("Before using trim\n");
      System.out.println(str);
      System.out.println("After using trim");
      System.out.println(str.trim() );
  }
}
```

In Listing 21, a String object, `str` is created and initialized with some value having few leading white spaces. The `trim()` method is then applied on str. Figure 25 shows the output:



▲ **Figure 25: Displaying the Output of the Trim Class**

Implementing the charAt() Method

The `charAt()` method returns the character located at a specified index. This method accepts an integer value, which represents an index number and returns the character present at that index number. The following code snippet shows the format of the charAt() method:

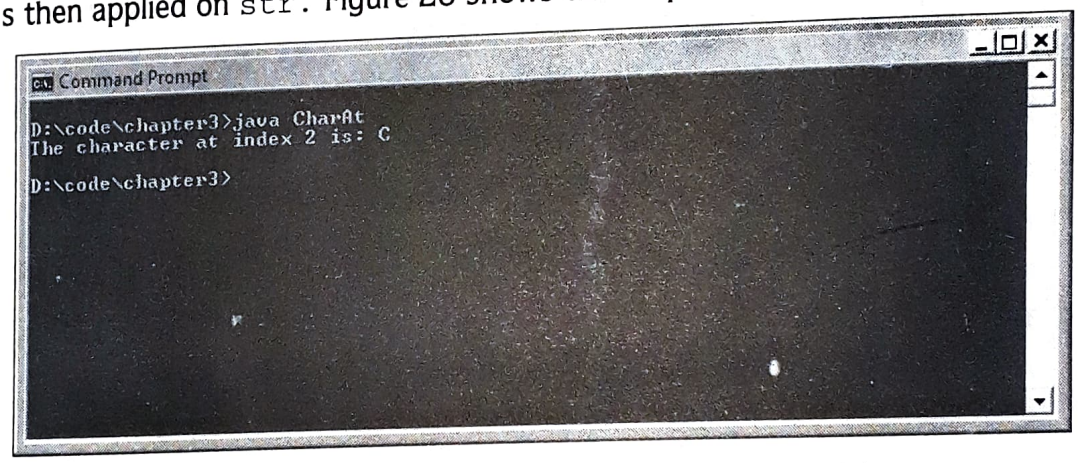
```
char charAt(int index);
```

Listing 22 demonstrates the implementation of the charAt() method (you can find the CharAt.java file on the CD in the code\chapter3 folder):

▶ Listing 22: Using the charAt() Method

```
public class CharAt {
  public static void main(String args[]) {
      String str = "B C D E ";
      System.out.println("The character at index 2 is: " + str.charAt(2));
  }
}
```

In Listing 22, a string `str` is created and initialized with some characters. The charAt() method is then applied on `str`. Figure 26 shows the output of Listing 22:



▲ Figure 26: Displaying the Output of the CharAt() Class

Implementing the getChars() Method

This method copies the characters from the current string to a destination character array. The character in the source string is read from the beginning character to end-1 position. They are copied into the destination array from its beginning position. The following code snippet shows the signature of the getChars() method:

```
void   getChars(int start_pos, int end_pos, char ch[],begin_pos);
```

In the preceding code snippet:

- **start_pos** represents the starting index of the character in the invoking string.
- **end_pos** represents the end index of the character in the invoking string.
- **ch[]** represents the character array in which the characters taken from the invoking strings are copied.
- **begin_pos** represents the beginning position of the array. It must be 0.

Listing 23 shows the implementation of the getChars() method (you can find the GetChar.java file on the CD in the code\chapter3 folder):

▶ **Listing 23: Using the getChars() Method**

```
public class GetChar {
  public static void main(String args[]) {
      String str ="Silly Silly Things cannot Disturb me ";
      int start_pos = 5;
      int end_pos = 25;
      char ch[] = new char[end_pos-start_pos];
      str.getChars(start_pos,end_pos,ch,0);
      System.out.print("copied characters are:" );
      System.out.print(ch);
  }
}
```

In Listing 23, first a string `str` is created and initialized with some texts. After that, two integer variables are created to specify the `start` and `end` position of the characters in the invoking string. Next, a character array, `ch` is created. At last, the `getChars()` method is applied on `str`. Figure 27 shows the output of the Listing 23:

▲ Figure 27: Displaying the Output of the GetChar() Class

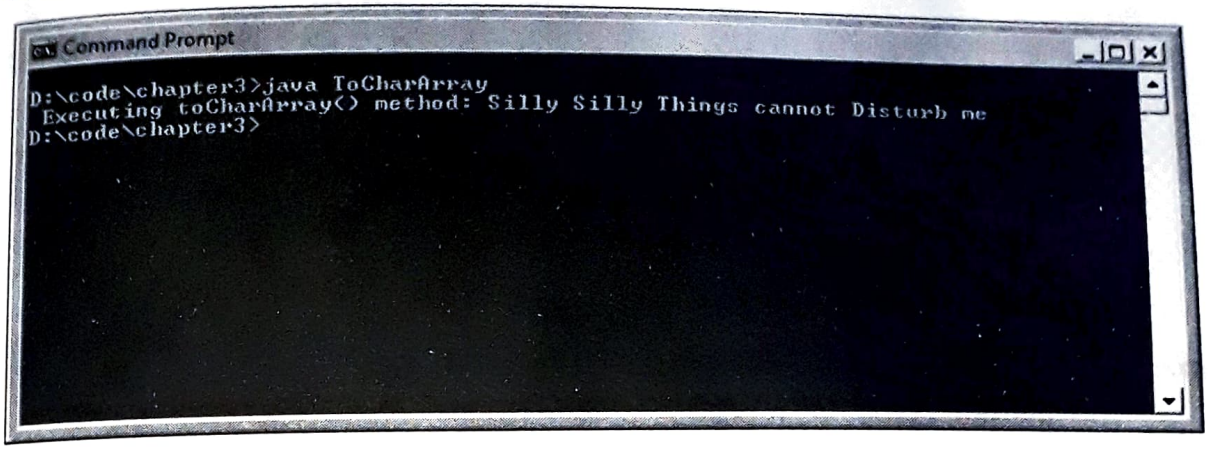
Implementing the toCharArray() Method

This method converts the invoking string into a character array. Listing 24 demonstrates the implementation of the toCharArray() method (you can find the ToCharArray.java file on the CD in the code\chapter3 folder):

▶ **Listing 24: Using the toCharArray() Method**

```
public class ToCharArray {
  public static void main(String args[]) {
      String str = "Silly Silly Things cannot Disturb me ";
      char ch[] = new char[str.length()];
      ch = str.toCharArray();
      System.out.print(" Executing toCharArray() method: ");
      System.out.print(ch);
  }
}
```

In Listing 24, first a string `str` is created and initialized with some texts. Next, a character array, `ch` of the length of `str` is created. At last, the toCharArray() method is applied on str. Figure 28 shows the output of Listing 24:



▲ **Figure 28: Displaying the output of the ToCharArray Class**

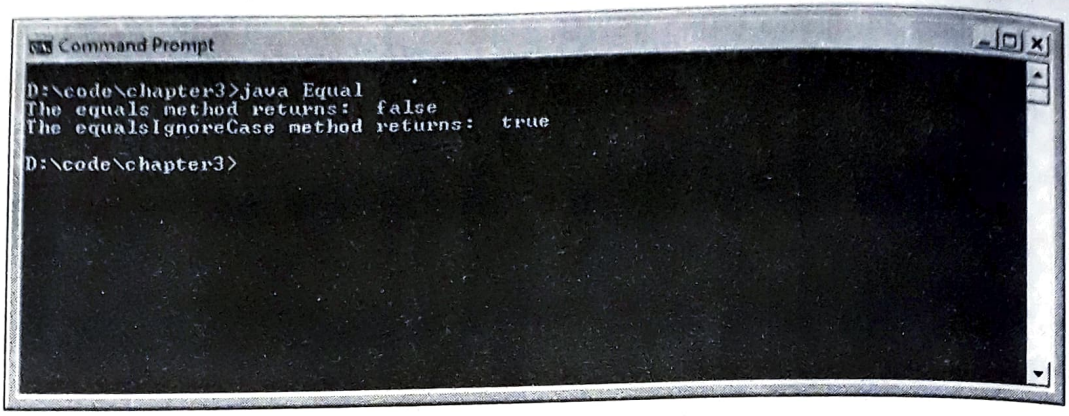
Implementing the equals() and equalsIgnoreCase() Methods

The `equal( )` and `equalIgnoreCase( )` methods are used for comparing the two strings. The String class overrides the `equal( )` method of the `Object` class. This method is implemented by comparing the positions of the individual characters in a string. The equalIgnoreCase( ) method does the same but ignores the case of characters. Both these methods return true if the strings are same else returns false. Listing 25 shows the implementation of the `equals()` and `equalsIgnoreCase()` methods (you can find the Equal.java file on the CD in the code\chapter3 folder):

▶ **Listing 25: Using the equals() and equalsIgnoreCase() Methods**

```
public class Equal
{
 public static void main(String args[])
 {
    String str = "Java is an Object oriented language";
    String str1 = "JAVA IS AN OBJECT ORIENTED LANGUAGE";
    boolean b1 = str.equals(str1);
    boolean b2 = str.equalsIgnoreCase(str1);
    System.out.println("The equals method returns:  " + b1);
    System.out.println("The equalsIgnoreCase method returns:  " + b2);
 }

}
```

In Listing 25, two strings `str` and `str1` are created and initialized with the same texts. The text in str is in lower case while the text contained by str1 has upper casing. Next, `equals()` and equalsIgnoreCase() methods are applied on `str` and `str1`. Figure 29 shows the output of Listing 25:



▲ Figure 29: Displaying the Output of the Equal Class

Implementing the regionMatches() Method

This method compares a portion of two strings whether they are equal or not. It returns true if the regions are same else returns false. Table 9 lists all the forms of the regionMatches() method:

Table 9: Forms of the regionMatches() Method

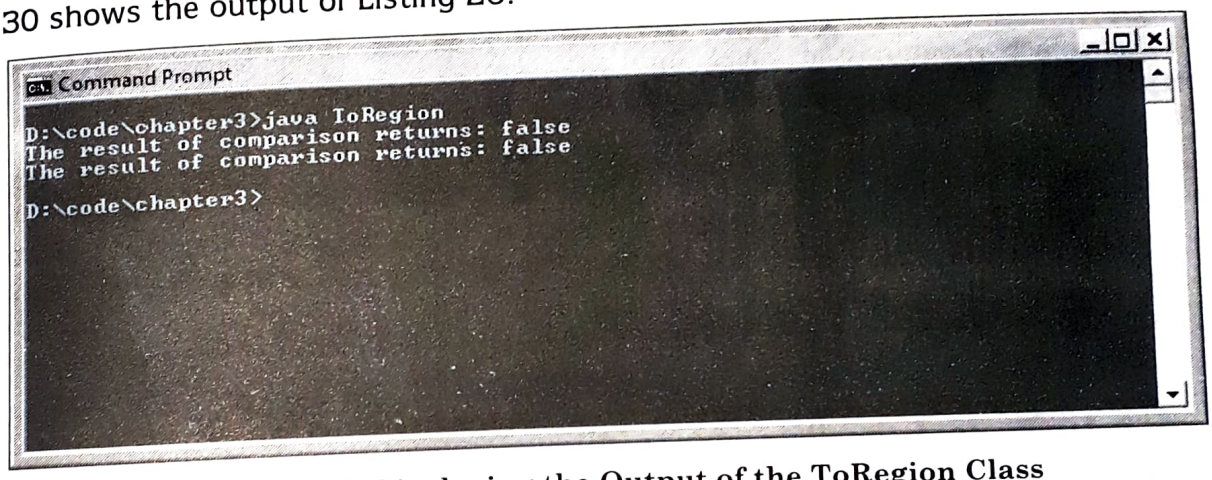
Method	Explanation
`boolean regionMatches(int start_pos_1, String str2, int start_pos_2, int len);`	Compares the characters of the invoking string with the total number of the characters in the second string. Here: ▪ **start_pos_1** represents the starting index of the characters in the invoking string. ▪ **str2** represents the second string. ▪ **start_pos_2** represents the starting index of the characters in the second string. ▪ **len** represents the total number of the characters in the second string that have to be compared with the characters of the invoking string.
`boolean regionMatches(boolean Ignore_Case, int start_pos_1, String str2, int start_pos_2, int len);`	This form of the regionMatches () method is similar to its first form except that it offers the facility to explicitly specify that whether the characters cases should be ignored or not. Here: ▪ **boolean Ignore_Case** represents the casing. If this parameter is set to true then the characters are not compared for casing, else compared. ▪ **start_pos_1** represents the starting index of the characters in the invoking string. ▪ **str2** represents the second string. ▪ **start_pos_2** represents the starting index of the characters in the second string. ▪ **len** represents the total number of the characters in the second string that have to be compared with the characters of the invoking string.

Listing 26 shows the implementation of both the forms of the regionMatches() method (you can find the ToRegion.java file on the CD in the code/chapter3 folder):

▶ **Listing 26: Using the regionMatches() Method**

```
public class ToRegion
{
 public static void main(String args[])
 {
     String str = "Hello Java ";
     String str1 = "HELLO HI HOW ARE You?";
     boolean b = str.regionMatches(4, str1, 7,2);
     System.out.println("The result of comparison returns: " + b);
     boolean c= str.regionMatches(true,4, str1, 7,2);
     System.out.println("The result of comparison returns: " + c);

 }
}
```

In Listing 26, two strings `str` and `str1` are created and initialized with some texts. The text in `str` is in lower case while the text contained by `str1` has mixed casing. Next, both the forms of the regionMatches() method that are listed in Table 9 are applied on `str` and `str1`. Figure 30 shows the output of Listing 26:



▲ Figure 30: Displaying the Output of the ToRegion Class

Implementing the startsWith() and endsWith() Methods

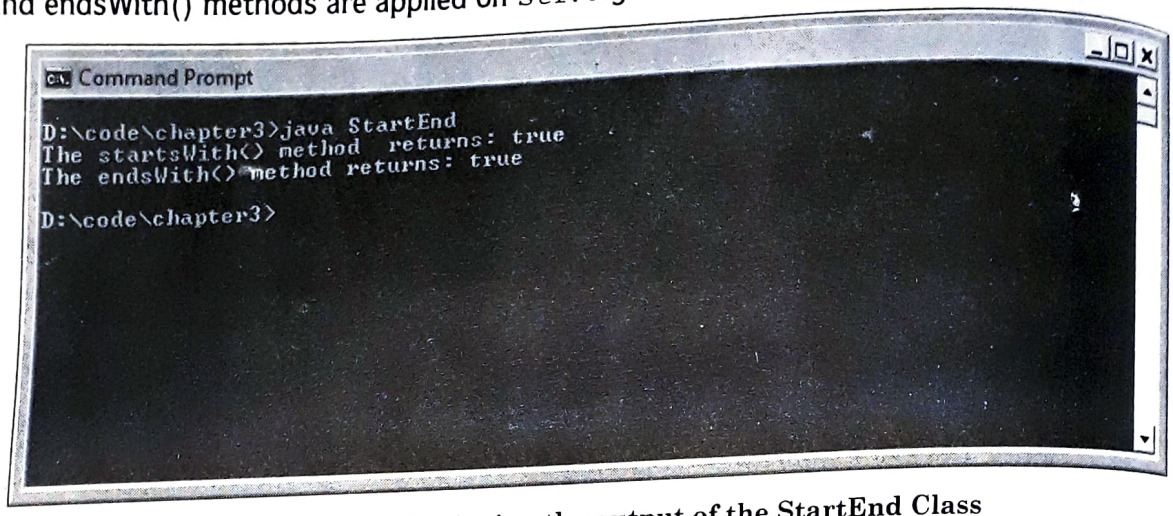
The `startsWith()` method accepts a string value and compares that value with the invoking string to test whether the invoking string starts with given string or not. It returns true if the invoking string starts with the specified string value, else returns false. The `endsWith()` method is the vice-versa of the startsWith() method. Listing 27 shows the implementation of the startsWith() and the endsWith() methods (you can find the StartEnd.java file on the CD in the code\chapter3 folder):

▶ **Listing 27: Using the startsWith() and the endsWith() Methods**

```
public class StartEnd{
 public static void main(String args[])
 {
     String str = "silly silly things cannot disturb me";
     boolean b = str.startsWith("silly");
     System.out.println("The startsWith() method  returns: " + b);
```

```
        boolean c = str.endsWith("me");
        System.out.println("The endsWith() method returns: " + c);
    }
}
```

In Listing 27, a string `str` is created and initialized with some texts. Next, the startsWith() and endsWith() methods are applied on `str`. Figure 31 shows the output of Listing 27:



▲ Figure 31: Displaying the output of the StartEnd Class

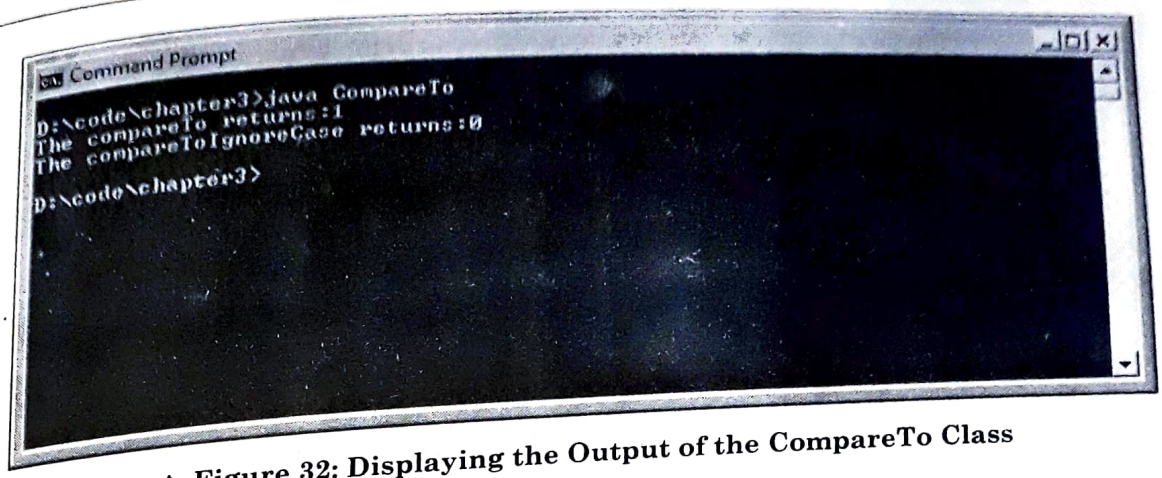
Implementing the compareTo() and compareToIgnoreCase() Methods

The `compareTo()` method compares two strings and check whether they are equal or not. The `compareToIgnoreCase()` method works similar to the `compareTo()` method but it ignores the casing of the strings. Both these methods return 0 when both the strings are equal, a value less than 0 when the string passed in any of the method is greater than the invoking string, or a value greater than 0 when the string passed in any of the method is less than the invoking string. Listing 28 shows the implementation of these methods (you can find the CompareTo.java file on the CD in the code/chapter3 folder):

▶ **Listing 28: Using the compareTo() and compareToIgnoreCase() Methods**

```
public class CompareTo{
  public static void main(String args[])
  {
      String str = "your";
      int b = str.compareTo("you");
      System.out.println("The compareTo returns:" + b);
      int c = str.compareToIgnoreCase("your");
      System.out.println("The compareToIgnoreCase returns:" + c);
  }
}
```

In Listing 28, a string, `str` is created and initialized with the value `your`. Then, `str` is compared with the value `you` using the `compareTo()` method and its result is displayed. Next, the `str` is compared with the value `your` using the `compareToIgnoreCase()` method. Figure 32 displays the result of these methods:



▲ Figure 32: Displaying the Output of the CompareTo Class

Implementing the indexOf() and lastIndexOf() Methods

The `indexOf()` and the `lastIndexOf()` methods return the first and last index of the occurrence of the character or string passed in these methods. Table 10 lists the various forms of these methods:

Table 10: Forms of the indexOf() and lastIndexOf() Methods

Method	Explanation
`int indexOf(int char)`	Returns the first index of the given character in the invoking string. You can also pass the ASCII value of a character to search its index. For example, you can use `indexOf(65)` in place of `indexOf('A');`
`int indexOf(int char, int start_index)`	Works similar to the previous method but starts searching for the character index from the given index. For example, `indexOf('a', 5);` In preceding example, the index of the character `a` is searched from the 5^{th} index of the invoking string.
`int indexOf(String str)`	Returns the first index of the given string in the invoking string.
`int indexOf(String str, int start_index)`	Works similar to the previous method but start searching for the string index from the given index. For example, `indexOf("abc", 5);` In preceding example, the index of the string "abc" is searched from the 5^{th} index of the invoking string.
`int lastIndexOf(int char)`	Returns the last index of the given character in the invoking string. You can also pass the ASCII value of a character to search its index.
`int lastIndexOf(int char, int start_index)`	Works similar to the previous method but starts searching for the character index from the given index.
`int lastIndexOf(String str)`	Returns the last index of the given string in the invoking string.
`int lastIndexOf(String str, int start_index)`	Works similar to the previous method but starts searching for the string index from the given index.

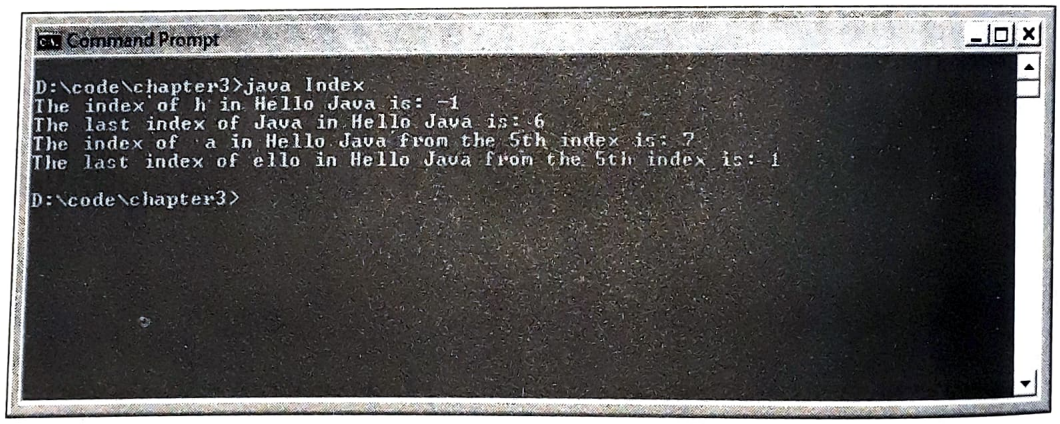
If a given character or string does not found in the invoking string then all the methods listed in Table 10 return -1.

Listing 29 shows the implementation of few of the commonly used methods listed in Table 10 (you can find the Index.java file on the CD in the code\chapter3 folder):

▶ Listing 29: Using the indexOf() and lastIndexOf() Methods

```
public class Index{
  public static void main(String args[]) {
      String str = "Hello Java";
      int b = str.indexOf('h');
      System.out.println("The index of h in Hello Java is: " + b);
      int c = str.lastIndexOf("Java");
      System.out.println("The last index of Java in Hello Java is: " +
      c);
      int d = str.indexOf('a',5);
      System.out.println("The index of  a in Hello Java from the 5th
      index is: " + d);
      int e = str.lastIndexOf("ello",5);
      System.out.println("The last index of ello in Hello Java from the
      5th index is: " + e);
  }
}
```

In Listing 29, a string, `str` is created and initialized with the value `Hello Java`. Next, various forms of the `indexOf()` and `lastIndexOf()` methods are applied on this string. Figure 33 shows the output of Listing 29:



▲ Figure 33: Displaying the Output of the Index Class

Implementing the toString() Method

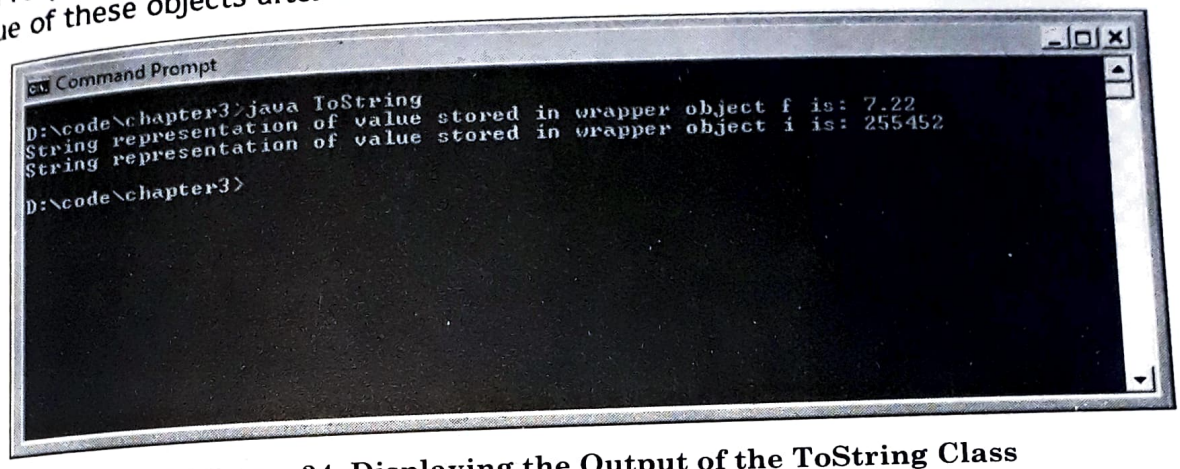
In Java, wrapper objects can be converted into string values using the toString() method. This method returns the same value stored in the wrapper object but that value can now be treated as a string. The converted string value can be incorporated with other strings. Listing 30 shows the implementation of the toString() method (you can find the ToString.java file on the CD in the code/chapter3 folder):

▶ Listing 30: Using the toString() Method

```
public class ToString {
  public static void main(String args[]) {
      Float f = new Float(7.22);
```

```
        system.out.println("String representation of value stored in
        wrapper object f is: "+ f.toString() );
        Integer i = new Integer(255452);
        System.out.println("String representation of value stored in
        wrapper object i is: " + i.toString());

    }
}
```

In Listing 30, two wrapper objects `f` and `i` of the wrapper classes `Float` and `Integer` are created respectively. Next, the toString() method is applied on these objects. Figure 34 shows the value of these objects after their conversion into strings:



▲ **Figure 34: Displaying the Output of the ToString Class**

Implementing the valueOf() Method

This method returns the string representation of the value passed in it. The value passed in this method can be of any primitive data type. After converting the value into a string, the converted value can be used with other strings. Table 11 lists various forms of the valueOf() method:

Table 11: Forms of the valueOf() Method

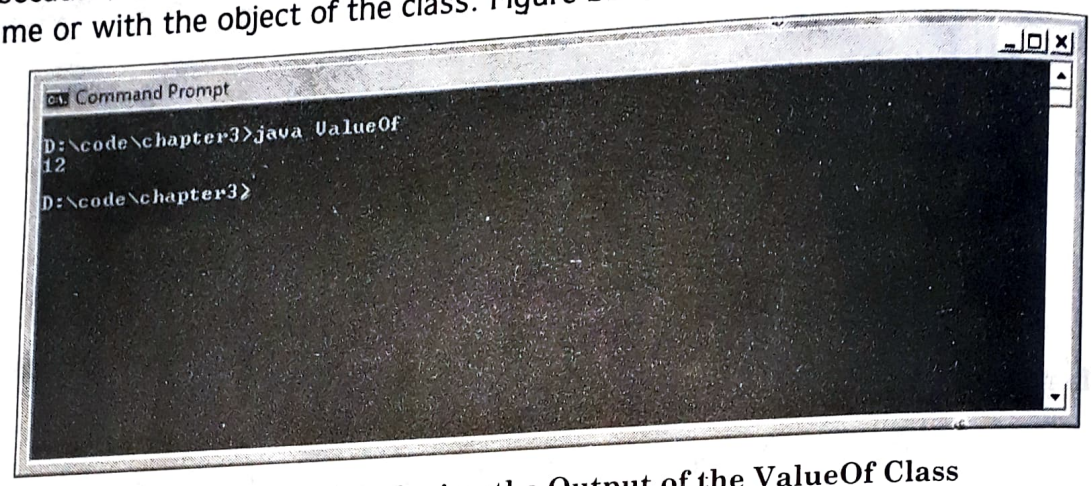
Method	Explanation
`static String valueOf(boolean b)`	Returns the string representation of the boolean value
`static String valueOf(char ch)`	Returns the string representation of the character value
`static String valueOf(char ch[])`	Returns the string representation of the given character array
`static String valueOf(int i)`	Returns the string representation of the integer value
`static String valueOf(long lon)`	Returns the string representation of the long value
`static String valueOf(Object o)`	Returns the string representation of the given object
`static String valueOf(double d)`	Returns the string representation of the double value
`static String valueOf(float f)`	Returns the string representation of the float value

Listing 31 shows the implementation of the most commonly used forms of the valueOf() method (you can find the ValueOf.java file on the CD in the code\chapter3 folder):

▶ **Listing 31: Using the valueOf() Method**

```
public class ValueOf{
  public static void main(String args[]) {
      System.out.println(String.valueOf(1) + String.valueOf(2));
  }
}
```

In Listing 31, the `valueOf()` method is applied on two integer values `1` and `2`. In Listing 31, you would have noticed that the class name, String is used with the valueOf() method. This is because, the valueOf() method is a static method and a static method is used with its class name or with the object of the class. Figure 35 shows the output of Listing 31:



▲ **Figure 35: Displaying the Output of the ValueOf Class**

With this, we have completed studying methods of the Sting class. Now, we compare the equals() method with the == operator.

TEST YOUR KNOWLEDGE

Q27. Write a program to create a string using the String class and display that string in reverse order.

Ans.

```
class TestYourKnowledge27
{
  public static void main(String args[])
  {
      String str=new String("Hello Java");
      int len=str.length();
      for(int x=len-1; x>=0; x--)
      {
         System.out.print(" " + str.charAt(x));
      }
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge27.java
D:\code\chapter3>java TestYourKnowledge27
```

Output:

```
a v a J o l l e H
```

Q28. **Write a program to create a string using the String class and check whether the string is a palindrome or not. A string is a palindrome that is spelled the same both forwards and backwards.**

Ans.

```
class TestYourKnowledge28 {
public static void main(String args[])
{
 String s="radar";
 int c=s.length();
 boolean flag=false;
 for(int x=0,y=c-1; x<c/2; x++, y--)
 {
     if(s.charAt(x)==s.charAt(y))
     {
         flag=true;
     }
     else
     {
         flag=false;
     }
 }
 if(flag==true)
 {
     System.out.println(" The string " + s + " is a palindrome");
 }
 else
 {
     System.out.println(" The string " + s + "is a not a palindrome");
 }
}
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge28.java
D:\code\chapter3>java TestYourKnowledge28
```

Output:

```
The string radar is a palindrome.
```

Q29. **Write a program to create a string and replace all the occurrence of an with the.**

Ans.

```
class TestYourKnowledge29{
 public static void main(String args[])
 {
     StringBuffer sb=new StringBuffer("He is an intelligent boy");
         for(int x=0; x<sb.length(); x++)
     {
     if(sb.charAt(x)== 'a' && sb.charAt(x+1) == 'n' &&
         sb.charAt(x+2)== 32)
```

```
            {
                sb.replace(x, x+2, "the");
            }

            }
            System.out.println("String after replacement is: " + sb);
        }
    }
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge29.java
D:\code\chapter3>java TestYourKnowledge29
```

Output:

```
String after replacement is: He is the intelligent boy.
```

Q30. Write a program to create a string array and sort all the strings contained by the array.

Ans.

```
class TestYourKnowledge30
{
 public static void main(String args[])
 {
    String str[]={"Hi", "Hello", "Java", "good"};
    String temp;
    for(int x=0; x<str.length; x++)
    {
        for(int y=x+1; y<str.length; y++)
        {
            if(str[y].compareToIgnoreCase(str[x])<0)
            {
                temp = str[x];
                str[x] =str[y];
                str[y] = temp;
            }
        }
    }

    for(int k=0; k<str.length;k++)
    {
        System.out.print(str[k] + " ");
    }
  }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge30.java
D:\code\chapter3>java TestYourKnowledge30
```

Output:

```
good Hello Hi Java
```

Q31. Write a program to create a string and convert all of its characters to upper case. Also display total number of blank spaces in that string.

Ans.

```
class TestYourKnowledge31
{
 public static void main(String args[])
 {
    String str="Today, environment is cool";
    String str1=new String();
    int c=0;
    str1=str.toUpperCase();
    System.out.println(str1);
    for(int x=0;x<str1.length(); x++)
    {
        if(str1.charAt(x)==32)
        {
            c++;
        }
    }
    System.out.println("Total numbers of spaces in string str1 are: "
        + c);

 }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge31.java
D:\code\chapter3>java TestYourKnowledge31
```

Output:

```
TODAY, ENVIRONMENT IS COOL
Total number of spaces in string str1 are: 3
```

Q32. Write a program to sort a list of students on the basis of their marks.

Ans.

```
// Program to sort a list of students on the basis of their marks
class TestYourKnowledge32
{
 public static void main(String args[])
 {
    int M[]={85,67,45,76};
    String SN[]={"NSJ","VDP","IIS","ABB"};
    int N=M.length;
    int I;
    String P=new String();
    for(I=0;I<=N-1;I++)
    {
        int big=M[I];
        int loc=I;
        for(int J=I+1;J<=N-1;J++)
        {
            if(big<M[J])
            {
                big=M[J];
```

```
                    loc=J;
                }
            }
            int temp=M[I];
            M[I]=M[loc];
            M[loc]=temp;
            P=SN[I];
            SN[I]=SN[loc];
            SN[loc]=P;
        }
        for(int k=0;k<=N-1;k++)
        {
            System.out.println(SN[k] );
        }
    }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge32.java
D:\code\chapter3>java TestYourKnowledge32
```

Output:

```
NSJ
ABB
VDP
IIS
```

Q33. Write a program to display student names on the basis of descending order of their marks.

Ans.

```
class TestYourKnowledge33
{
 public static void main(String args[])
 {
    int N,K,I,TEMP,BIG,LOC;
    int A[]={52,36,96,58,47,41};
    String SN[]={"AMEY","VINAYA","NIKHIL","RACHNA","PINKY","DHABBA"};
    N=A.length;
    String P=new String();
    for(K=0;K<=N-1;K++)
    {
        BIG=A[K];
        LOC=K;
        for(I=K;I<=N-1;I++)
        if(BIG<A[I])
        {
            BIG=A[I];
            LOC=I;
        }
        TEMP=A[K];
        A[K]=A[LOC];
        A[LOC]=(TEMP);
        P=SN[K];
        SN[K]=SN[LOC];
        SN[LOC]=P;
    }
    System.out.println("RESULTS");
```

```
        for(I=0;I<=N-1;I++)
        system.out.println(" "+SN[I]+" "+A[I]);
    }
}
```

Execution:

```
D:\code\chapter3>javac TestYourKnowledge33.java
D:\code\chapter3>java TestYourKnowledge33
```

Output:

```
RESULTS
NIKHIL 96
RACHNA 58
AMEY 52
PINKY 47
DHABBA 41
VINAYA 36
```

Comparing the equals() Method and == Operator

The `equals()` method is used to compare objects while the == operator is used to compare the equality of primitive data values. Both of them return true if the objects or values are equal, else return false. Implementation of the equals() method has already been given in Listing 25. Listing 32 shows the implementation of the == operator (you can find the EqualOperator.java file on the CD in the code\chapter3 folder):

▶ **Listing 32: Using the == Operator**

```
public class EqualOperator{
  public static void main(String args[]) {
      int val = 2000;
      System.out.println(val % 2 == 0);
      System.out.println(val == 0);
  }
}
```

In Listing 32, a variable `val` of type `int` is declared and initialized with the value 2000. The == operator is then implemented on `val`. Figure 36 shows the output of Listing 32:

▲ **Figure 36: Displaying the Output of the EqualOperator Class**

Using the StringBuffer Class

Similar to the String class, the StringBuffer is also present in the java.lang package. The difference between both the classes is that the String class creates immutable strings while the StringBuffer class helps in creating mutable strings. In other words, the strings created using the StringBuffer class can be changed dynamically. Table 12 lists the constructors of the StringBuffer class:

Table 12: Constructors of the StringBuffer Class

Constructor	Explanation
`StringBuffer()`	Creates a string object without any value. The initial capacity for this constructor is 16 characters. For Example: `StringBuffer mystr = new StringBuffer();`
`StringBuffer( int length)`	Creates a string object without any value. However, this constructor accepts an integer value that specifies the size of the string object. For example: `StringBuffer sb = new StringBuffer(10);`
`StringBuffer(String str)`	Creates a string object whose size is equal to the size of the string passed in this constructor. For example: StringBuffer mystr = new StringBuffer("Test");

Table 13 lists the methods of the StringBuffer class:

Table 13: Methods of the StringBuffer Class

Method	Explanation
`StringBuffer append(Object ob)`	Appends the value stored in the given object to the end of the invoking StringBuffer object.
`StringBuffer append(boolean b)`	Appends a boolean value at the end of the invoking StringBuffer object.
`StringBuffer append(char ch)`	Appends a character at the end of the invoking StringBuffer object.
`StringBuffer append(char ch[])`	Appends the value contained by a character array at the end of the invoking StringBuffer object.
`StringBuffer append(char ch[], int start_index, int length)`	Appends a portion of the character array at the end of the invoking StringBuffer object. The argument Start_index specifies the index of the first character to append. The argument length specifies the total number of characters that have to be appended in the invoking StringBuffer object.
`StringBuffer append(double d)`	Appends a double value at the end of the invoking StringBuffer object.
`StringBuffer append(float f)`	Appends a float value at the end of the invoking StringBuffer object.
`StringBuffer append(int i)`	Appends an integer value at the end of the invoking StringBuffer object.

Table 13: Methods of the StringBuffer Class

Method	Explanation
`StringBuffer append(long l)`	Appends a long value at the end of the invoking StringBuffer object.
`StringBuffer append(String str)`	Appends a string at the end of the invoking StringBuffer object.
`int capacity()`	Returns the total number of characters that an invoking StringBuffer object can contain.
`char charAt(int index)`	Returns the character located at the given index in the invoking StringBuffer object.
`StringBuffer delete(int start_index, int end_index)`	Removes a sequence of characters from the invoking StringBuffer object. The start_index specifies the index of the starting character and the end_index specifies the index of the last character.
`StringBuffer deleteCharAt(int index)`	Removes a character located at the given index in the invoking StringBuffer object.
`void ensureCapacity (int capacity)`	Ensures that the capacity of the invoking StringBuffer object is equal to the specified capacity. If the capacity of the invoking object is less than the specified capacity then the size of the object increases automatically.
`void getChars(int start_index, int end_index, char ch[], int ch_start_index)`	Copies a sequence of characters of the invoking object to the specified character array. The characters to be copied come between start_index and end_index. The parameter ch[] is the character array in which characters of the invoking object will be copied. The parameter ch_start_index is the index in the character array from where the coping of characters will start.
`StringBuffer insert(int index, boolean b)`	Adds a boolean value at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, char c)`	Adds a character at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, char ch[])`	Adds characters of the specified array at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, char ch[], int start_index, int length)`	Adds a portion of the specified array at the specific index in the invoking StringBuffer object.
`StringBuffer insert(int index, double d)`	Adds a double value at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, float f)`	Adds a float value at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, int i)`	Adds an integer value at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, long l)`	Adds a long value at the specified index in the invoking StringBuffer object.

Table 13: Methods of the StringBuffer Class

Method	Explanation
`StringBuffer insert(int index, Object ob)`	Adds an object at the specified index in the invoking StringBuffer object.
`StringBuffer insert(int index, String str)`	Adds a string at the specified index in the invoking StringBuffer object.
`int length()`	Returns the length of the invoking StringBuffer object.
`StringBuffer replace( int start_index, int end_index, String str)`	Replaces the characters between given range with the specified string.
`StringBuffer reverse()`	Reverses the characters sequence of the invoking StringBuffer object.
`void setChatAt(int index, char ch)`	Sets the given character at the specified index in the invoking StringBuffer object.
`void setLength(int length)`	Sets the length of the invoking StringBuffer object.
`String substring(int start)`	Returns a substring from the invoking StringBuffer object, which starts from the specified index.
`String substring(int start_index, int end_index)`	Returns a substring from the invoking StringBuffer object, which comes between the given ranges.
`String toString()`	Returns the value of the StringBuffer object as a string.

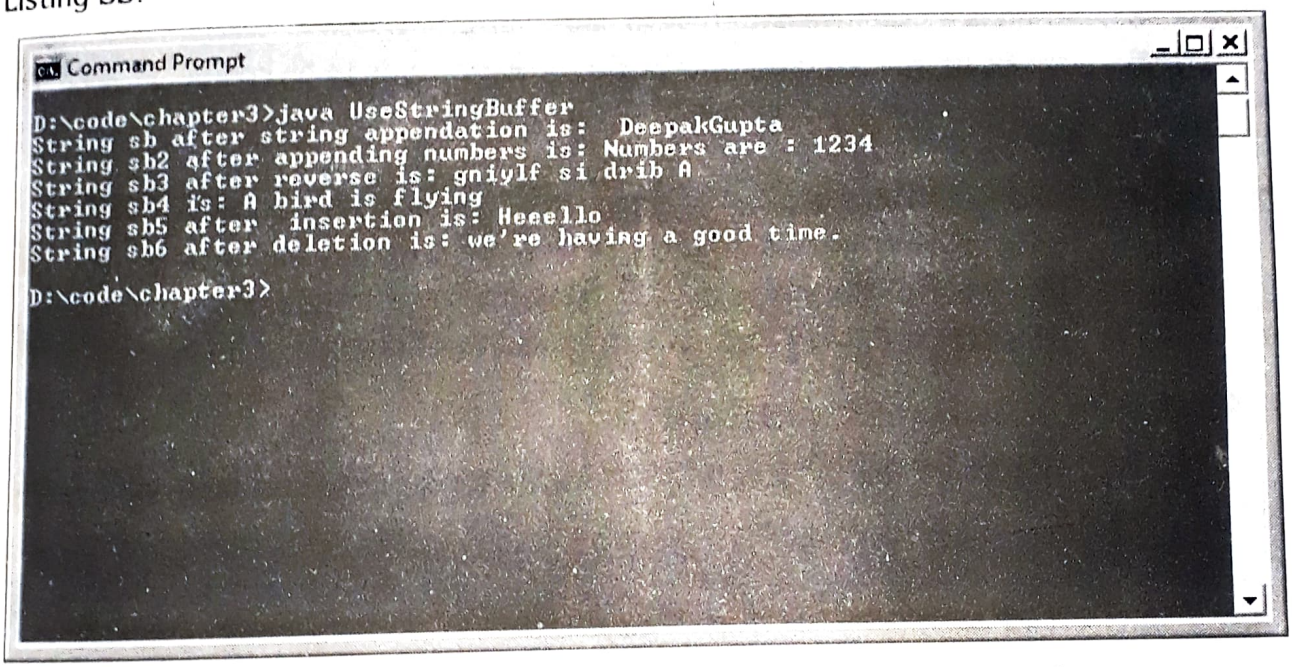
Listing 33 shows the implementation of few of the methods listed in Table 13 (you can find the UseStringBuffer.java file on the CD in the code\chapter3 folder):

▶ **Listing 33: Using Methods of the StringBuffer Class**

```
class UseStringBuffer{

  public static void main(String args[]) {
// Creating first string object
StringBuffer sb = new StringBuffer("Deepak");
// appending String
sb.append("Gupta");
System.out.println("String sb after string appendation is:  " + sb);

// Creating second string object
StringBuffer sb2 = new StringBuffer("Numbers are : ");
// appending numbers
sb2.append(1234);
System.out.println("String sb2 after appending numbers is: "+ sb2);
// Creating third string object
StringBuffer sb3 = new StringBuffer("A bird is flying");
// Reversing the string object
sb3.reverse();
System.out.println("String sb3 after reverse is: " + sb3);

// Creating fourth string object
StringBuffer sb4 = new StringBuffer("A bird is flying");
// Applying toString() on string object
System.out.println( "String sb4 is: " + sb4.toString() );
```

```
// creating fifth string object
StringBuffer sb5 = new StringBuffer("Hello");
sb5.insert(1, "ee");
System.out.println("String sb5 after  insertion is: " + sb5);

// creating sixth string object
StringBuffer sb6 = new StringBuffer("we're not having a good time.");
sb6.delete(5, 9);
System.out.println("String sb6 after deletion is: " + sb6);
}
}
```

In Listing 33, StringBuffer objects `sb`, `sb2`, `sb3`, `sb4`, `sb5`, and `sb6` are created. Then, the `append()` method is applied on `sb` and `sb2`, which is followed by applying reverse method on `sb3`. Next, the `toString()` method is applied on `sb4` and the `insert()` method is applied on `sb5`. In last, the `delete()` method is applied on `sb6`. Figure 37 shows the result of Listing 33:



▲ **Figure 37: Displaying the Output of the UseStringBuffer Class**

This is all about using the StringBuffer class. We have also reached at the end of this chapter. Therefore, let's have a summary of the topics covered in this chapter.

Summary

This chapter has been started with the discussion on arrays. Various types of arrays and the processes to create and use arrays are elaborated in this chapter. The discussion moves ahead by depicting the Vector class, a Java class used to create dynamic array. A vector can contain only objects; therefore, wrapper classes are discussed next, to convert primitive data types to objects. After wrapper classes, the String class, and its methods are discussed. A String class can only create immutable strings but sometimes you may need to modify a string to get the desired information. Therefore, the StringBuffer class is depicted at the end of this chapter.

Review Questions

Here, let's answer the following types of questions:

- ❒ True or False
- ❒ Multiple Choice
- ❒ Short Answers
- ❒ Debugging

■ True or False

1. An array can only store elements of similar data types. **True**
2. An array of objects cannot be created in Java. **False**
3. The Vector class is used to create dynamic arrays. **True**
4. The methods of Enumeration interface are used to iterate the elements of a vector. **True**
5. String is a primitive data type in Java. **False**
6. To use strings, an object of the String class needs to be created in Java. **True**
7. An array of strings can also be created in Java. **True**
8. The length attribute is used to calculate the length of a string. **False**
9. A string can only be concatenated with another string. **False**
10. The length attribute is used to calculate the length of an array. **True**
11. The StringBuffer class is used to create mutable strings. **True**
12. The Wrapper class is used to create objects of primitive data types. **True**

■ Multiple Choice Questions

Q1. Which of the following is a static method of the java.lang.String class?

A. append()
B. concat()
C. insert()
D. valueOf()

Ans. Option D is correct.

Q2. Which of the following method of the java.lang.Integer class returns a primitive int type?

A. intValue()
B. parseInt()
C. toString()
D. valueOf()

Ans. Option B is correct.

Q3. Which of the following can be used to concatenate strings?

A. concat()
B. +
C. ==
D. =

Ans. Options A and B are correct.

Q4. Which of the following method is used to calculate the length of a string?

A. valueOf()
B. length
C. length()
D. indexOf()

Ans. Option C is correct.

Q5. **Which of the following is not a Wrapper class?**

A. Byte
B. Vector
C. Short
D. Double

Ans. Option B is correct.

Q6. **Which of the following class is used to create a dynamic array?**

A. String
B. Vector
C. StringBuffer
D. Double

Ans. Option B is correct.

Q7. **Which of the following is a valid one-dimensional integer array declaration and creation?**

A. int x[ ]= new int[arr_size];
B. int [ ] x= int[arr_size];
C. int x[ ]= int[arr_size];
D. int x[ ]= new int[ ];

Ans. Option A is correct.

Q8. **Which of the following is a valid multi-dimensional integer array declaration and creation?**

A. int x[ ][ ]= new int[col][row];
B. int [ ] [ ] x= int[row][col];
C. int x[ ] [ ] = new int[row][col];
D. int x[ ][ ] = new int[row];

Ans. Option C is correct.

Q9. **Which of the following is a valid uneven multi-dimensional array declaration and creation?**

A. int x[ ][ ]= new int[col][ ];
B. int [ ] [ ] x= int[row][col];
C. int x[ ] [ ] = new int[row][col];
D. int x[ ][ ] = new int[row][ ];

Ans. Option D is correct.

Q10. **Which of the following method is used to copy array elements to another array?**

A. System.arraycopy();
B. toString();
C. replace();
D. startsWith();

Ans. Option A is correct.

Q11. **Which of the following method is used to trim strings?**

A. valueOf();
B. toString();
C. trim();
D. charAt();

Ans. Option C is correct.

Q12. **Which of the following method returns a portion of the invoking string?**

A. substring();
B. toString();
C. replace();
D. toCharArray();

Ans. Option A is correct.

■ Short Answer Questions

Q1. What is an array and why we need an array?

Ans. An array is a collection of data storage locations, each of which holds the same type of data. There can be a situation where you need to store similar type of values for a large number of data items. For example, to store telephone numbers of 50 persons, normally you have to declare 50 variables each storing a number in it, which is quite a tedious and complex approach.

With arrays, we can declare a single variable that is capable of storing all the 50 numbers. The advantage of using arrays is that they simplify programming by replacing a number of statements with just one or two statements.

Q2. Write down the statements to declare and create a one-dimensional array that can store 5 elements of integer type.

Ans. An array of integer type to store 5 elements can be declared and created as follows:

```
int store[ ];
store = new int[5];
```

These steps can be combined in one step as follows:

```
int  store[ ]=new int[5];
```

Q3. Write down a statement to declare, create, and initialize a one-dimensional integer array without using the new operator. That array should contain 5 random numbers.

Ans. In Java, an array without using the new operator can be declared, created, and initialized as follows:

```
int val[ ] = {1,25,3,14,5};
```

Q4. Is there any method in Java to copy elements of an array to another array? If yes then give the syntax to use that method.

Ans. The `System.arraycopy()` is a method of System class, which is used to copy an array elements from one array to another. The benefit of using this method is that you can copy the whole array into another array and array elements within a specified range as well. This method also saves your time that you spend in creating a long and complex program to copy an array into another array. The syntax to use `System.arraycopy()` method is:

```
System.arraycopy(Object first_array, int start_position, Object
    second_array, int start_pos2, int num_element);
```

In the preceding syntax:

- `System.arraycopy` is the method, which is used to copy one array into another array.
- `Object first_array` is the name of the first array.
- `int start_position` is the starting position in the first array from where the elements of the first array will be copied to the second array.

- `Object second_array` is the name of the second array.
- `int start_pos2` is the starting position in the second array. It is the position from where the element of the first array starts copying. If any of the second array elements will be present at that location then, it will be overridden by elements of the first array.
- `int num_element` represents total number of first array elements that are to be copied into second array.

Q5. Define the Vector class.

Ans. The Vector class is defined in the java.util package. The Vector class implements a dynamic array called vector. A vector always has some initial capacity to store elements. When this initial capacity is reached then the size of the vector automatically increases. The following are the advantages of using vector over arrays:

- A vector can contain heterogeneous objects.
- You can add or remove objects from a vector according to the requirements.

Q6. Define Wrapper classes in Java and give their types.

Ans. In Java, there are eight primitive data types: byte, short, int, long, float, double, boolean, and char. These primitive data types cannot be treated as objects until you will not convert them into objects using their respective Wrapper class. By converting primitive data types into their respective objects, Wrapper classes serve the following two purposes:

- After conversion into objects, the primitive data types can be used in vectors.
- The wrapper classes provide a variety of utility methods to convert any of the object into primitive types (such as byte, int, float, boolean, and char) or vice versa.

The following are the Wrapper classes available in Java:

- Character
- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean

Q7. Which of the Java class allows you to work with strings? Can an array to store multiple stings, be created? If yes, then create an array, str to store 5 strings.

Ans. In Java, the String class allows us to work with strings. To work with strings in Java, we have to create an object of the String class as follows:

```
String str= new String();
```

The preceding statement creates an object, str of the String class. Now, we can assign any string value to str as follows:

```
str=" Hello Java";
```

This is how strings can be created in Java. Apart from this, we can also create an array of strings. An array containing 5 strings can be created as follows:

```
String str[]=new String [5];
```

The preceding statement creates a string array, str, which can contain 5 strings.

Q8. List all the ways to concatenate strings in Java.

Ans. In Java, strings can be concatenated using the following two ways:

- Using the concatenation operator, +
- Using the concat() method of the String class

Q9. Consider the following string object:

```
String str=new String("Java is a robust programming language");
```

Which of the String class method can be used to calculate the length of str?

Ans. The length of a string can be calculated using the length() method of the String class.

Q10. Is there any method in Java, which returns a portion of the invoking string? If yes, then describe that method.

Ans. In Java, the `substring()` method returns substring of the original String. The `substring()` method is available in the following two forms:

- substring(int begin)
- substring(int begin, int end)

One form of the `substring()` function takes a single parameter, which creates a new String that begins at the index specified by the argument value. The following code snippet shows the use of the `substring(int begin)` method:

```
String str = "EDUCATION";
System.out.println (str.substring(4));
```

The preceding code snippet prints the value ATION. The second form of `substring()` method takes two parameters. It creates a substring that starts at the index of the first parameter and ends at the index of the second parameter. The character at the start index is included in the substring, but the character at the end index is not. The following code snippet shows the use of the substring (int begin, int end) method:

```
String str = "EDUCATION";
System.out.println (str.substring(3,6));
```

The preceding code snippet prints the value CAT.

Q11. Compare the equals() method and the == operator.

Ans. The equals() method is used to compare objects while the == operator is used to compare the equality of primitive data values. Both of these return true if the objects or values are equal, else return false.

Q12. Which Java class contains the append() method and what purpose the append() method completes?

Ans. The append() method is a method of java.lang.StringBuffer class. This method is used to add a value to the end of the invoking string. The value it appends can be a primitive data type, object, or String.

Debugging Exercises

Q1. What will be the output of the following program?

```
public class Test1 {
    public static void main(String args[]) {
        int x[]={5,2.5,6,7.8,9};
        for(int k=0; k<x.length; k++) {
            System.out.println(x[k]);
        }
    }
}
```

Ans. The program will not compile because the array x is of type int but it also contains the element of double type.

Q2. What will be the output of the following program?

```
public class Test2 {
    public static void main(String args[]) {
        int x[][]={{3,9},{2,5},{4,4}};
        for(int k=0; k<x.length; k++) {
            for(int j=0; j<x.length-1; j++) {
                System.out.println(x[k][j]+2);
            }
        }
    }
}
```

Ans. The program will print

5

11

4

7

6

6

Q3. What will be the output of the following program?

```
public class Test3 {
    public static void main (String args[]) {
    char arr[ ] = { 'H','E','L','L','O'};
    char arr1[ ] = new char[10];
```

```
            System.arraycopy(arr, 1, arr1,0, 2);
            System.out.print("Array after copy is: " );
            for( int y=0; y<arr1.length; y++)
            {
                System.out.print(arr1[y]);
            }
        }
       }
```

Ans. The program will print:

Array after copy is: EL

Q4. What will be the output of the following program?

```
import java.util.*;
class Test4 {
    public static void main(String args[]) {
        Vector vect = new Vector(2,5);
        System.out.println(vect.size());
        vect.addElement(new Boolean(false));
        vect.addElement(new Integer(10));
        vect.addElement(new Double(5.84));
        System.out.println(vect.capacity());
        System.out.println((Boolean)vect.firstElement());
    }
}
```

Ans. The program will print:

0

7

false

Q5. What will be the output of the following program?

```
class Test5 {
    public static void main(String args[]) {
        String str="Hello";
        for(int x=20; x<str.length()+18; x++) {
            System.out.println(str + "java");
        }
    }
}
```

Ans. The program will print:

Hellojava

Hellojava

Hellojava

Q6. What will be the output of the following program?

```
class Test6 {
    public static void main(String args[]) {
        String str[]={"Hi", "how", "are", "you"};
        for(int x=0; x<str.length(); x++) {
            System.out.println(str[x].length());
```

```
        }
    }
}
```

Ans. The program will not compile because the str.length() method used in the for loop is wrongly implemented. In for loop, the str.length() method is used to get the length of the string array, str, which is wrong. This is because, the length() method returns the length of a string while the length attribute returns the length of an array. Therefore, the str.length() method should be replaced with str.length to execute the program.

Q7. What will be the output of the following program?

```
public class Test7{
 public static void main(String args[]) {
    String str = "Book ";
    String str1 = "Test";
    boolean b = str.regionMatches(0, str1, 2,2);
    System.out.println(b);
 }
}
```

Ans. The program will print:

false

Q8. What will be the output of the following program?

```
public class Test8 {
 public static void main(String args[]) {
    String str = "Excellent ";
        for(int x=0; x<str.length; x++)
    {
        System.out.println(str);
    }
 }
}
```

Ans. The program will not compile because the str.length() method should be used to get the length of the string in place of str.length.

Q9. What will be the output of the following program?

```
public class Test9 {
 public static void main(String args[]) {
    String str = "Excellent";
        System.out.println(str.replace('t', ' '));
 }
}
```

Ans. The program will print Excellen.

Q10. What will be the output of the following program?

```
public class Test10 {
    public static void main(String args[]) {
        String str=new String("Hello");
        String str1=new String("Hello");
```

```
            int x=5;
            if(str.equals(str1)) {
                System.out.println("true 1");
            }
            if(str==x) {
                System.out.println("true 2");
            }
        }
    }
```

Ans. The program will not compile because the string value is compared with the int value.

Q11. What will be the output of the following program?

```
import java.io.*;
public class Test11 {
    public static void main(String args[]) {
        StringBuffer str=new StringBuffer("hello");
        System.out.println(str.reverse());
    }
}
```

Ans. The program will print:

olleh

Q12. What will be the output of the following program?

```
public class Test12 {
 public static void main(String args[]) {
    String org = "this";
    String search = "is";
    String sub = "was";
    String result = "";
    int i=org.length();
    for(int j=0;j<i;j++) {
        result = org.substring(0, j);
        result = result + sub;
        result = result + org.substring(j + search.length());
        org = result;
    }
 System.out.println(org);
}
}
```

Ans. The program will print:

Wwwwasis