02 Working with Java Members and Flow Control Statements

IN THIS CHAPTER

2.1 Explaining Data Types
2.2 Declaring Variables
2.3 Declaring Classes
2.4 Declaring Methods
2.5 Explaining Constructors
2.6 Creating Objects
2.7 Explaining Access Specifiers
2.8 Explaining Type Casting and Type Conversion
2.9 Implementing Flow Control Statements

A Java program comprises `Object Oriented Programming` (OOP) components, such as classes, variables, and methods which are known as Java members. A Java class is a template used to create Java programs consisting of variables (state) and methods (behavior). The variables are used to hold data items, whereas the methods are used to perform specific operations on those data items. Each object of a class has its own unique set of variables and methods. All these Java members are identified by a unique name, which is known as identifier in Java.

A Java program, similar to the programs in other programming languages, uses variables, data types, operators, and arrays to execute the program logic. A variable is the basic entity of a program, and can be defined as an arbitrary placeholder for storing data. A variable must be assigned a data type, which specifies the type of data that the variable can store. For example, a variable of the int data type can store integer values. To increase the cohesion of the Java program, methods are used to perform specific tasks. While declaring the members of a class, you can set its accessibility by using the public, private and protected keywords. These are access specifiers of the Java programming language.

The instructions in a program execute in a sequential manner, i.e. one after another in their hierarchy. Sometimes, you may need to change this sequential execution of your program to get the desired output. For this, Java provides some flow control statements that effectively changes the sequential control flow of the program.

In this chapter, you learn the data types available in Java programming language. The chapter also discusses how to declare a variable in a Java program. In addition, you learn about classes, methods and objects. This chapter also specifies the working nature of the public, protected, and private keywords. Towards the end of the chapter, you learn about the flow control statements in Java. Let's discuss data types now.

2.1 Explaining Data Types

Java is a strongly typed language, therefore it is important to understand the data types available in Java. You cannot even declare a variable without specifying its data type in Java program. The data types available in Java can be divided into four broad categories:

- `Integer`: Represents integers using byte, short, int, and long
- `Floating-point`: Represents fractional numbers using float and double data types
- `Character`: Represents `char`, which is used to store characters
- `Boolean`: Represents logical values using `boolean` data type

Let's start discussion with Integer data types first.

Integer Data Types

Integer data types are used to store integer data items. Java provides four integer data types to store the integer values — `byte`, `short`, `int`, and `long`. As Java only supports the signed integers, these integers types have `-ve` and `+ve` values. Among all, byte is the smallest integer data type while long is the largest `integer` data type. Table 1 lists the width and range of the `integer` types:

Table 1: Width and Range of Integer Data Types

Data Type Representation	Width (bytes)	Range
byte	1	-128 to 127
short	2	-32, 768 to 32, 767
int	4	-2,147,483,648 to 2, 147,483,647
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Few examples of integer data types are:

```
byte x;
short sum;
int age;
long milliseconds;
```

Here, x, sum, age, and milliseconds are variables of `byte`, `short`, `int`, and `long` datatype respectively. As shown in Table 1, the width of different types are different, therefore you must take care while using these data types. If you need to store a small data, for example, 25, you must use the `byte` data type as it is sufficient to store 25. If you use any other data type than `byte`, it allocates extra memory space unnecessarily which leads to increased execution time.

The long data type can also be declared by appending l or L at the end of the data. For example 125l or 125L, indicates that 125 is of long data type.

Floating-point Data Types

In Java, to manipulate the real numbers, the `floating-point` data type is used. Java provides two floating-point data types – float and double. The float data type is used to store the single precision numbers whereas the double data type is used to store the double precision numbers. The float and double data types are used to represent numbers along with decimal values. The `float` data type can represent a maximum of 7 digits after decimal while the double data type can represent 15 digits. Table 2 lists the width and range of the `floating-point` data types:

Table 2: Width and Range of Floating-point Data Types

Data Type	Width (bytes)	Range
float	4	--3.4e38 to –1.4e-45 for negative values and 1.4e-45 to 3.4e to 3.4e38 for positive value
double	8	–1.8e308 to –4.9e-324 for negative values and 4.9e-324 to 1.8e308 for positive values

An example of floating-point data types is as follows:

```
float x=3.458f;
double d=3.24;
```

In the preceding code snippet, x is declared and initialized as float data type, whereas d is declared as double type. Let's discuss these two data types in the following sections.

The float Data Type

The `float` data type is used to represent the `single-precision` values, which use 32 bits (4 byte) of the memory space. You are suggested to use `float` data type when you don't need to store a large degree of precesion. A `float` data type can store maximum 7 digits after the decimal point. While declaring and using `float` data type, you must add the `f` or `F` postfix to the data. If you declare and initialize a `float` data type variable without the `f` or `F` postfix, it causes the compilation error. As all the floating-point numbers are considerd as double in Java. For example, consider the code snippet:

```
float height=6.1;      //incorrect
float height=6.1f;    //correct
```

In the preceding code snippet, the first declaration is incorrect as `f` postfix is not added at the end of data, therefore it is considered as double.

The double Data Type

The double data type is used to represent the `double-precision` values, which use 64 bits (8 bytes) of the memory space. You are suggested to use double data type when you need to deal with itterative calculations, which results in a value of larger width. Although in Java, all the floating-point numbers are considered as double, you can use the d or D postfix with double type of data. It does not make any difference whether you are using the d or D postfix or not with the double data type. For example, consider the following code snippet:

```
double pi=3.14;
double pi=3.14d;
```

In the preceding code snippet, both the declarations are correct and represent the same double value. After discussing the two types of floating-point data types, let's discuss the character data type in the next section.

Character Data Type

The character data type is used to represent and store the character constants in Java programs. The character data type available in Java is `char`. This data type follows the Unicode character set to represent the characters that can be uniquely identified by the processor. Unicode is a standard character set that supports all the international characters of various languages, such as American, Latin, and so on. Table 3 lists the width and range of the character data type:

Table 3: Width and Range of Character Data Type

Data Type Representation	Width (bytes)	Range
`char`	2	0 to 65535

An example of `char` data types is as follows:

```
char grade='A';
```

In the preceding code snippet, the `grade` variable is declared and initialized with data `'A'`. You must remember to enclose any character within single quotes, while initializing a character data type.

The boolean Data Type

The `boolean` data type represents two logical values denoted by true or false. Let's see the folowing code snippet to initialize and use boolean data type:

```
boolean x = true;
```

In the preceding code snippet, the `x` variable of boolean data type is declared and initialized as true. You can also initialize it with false. All the data types discussed here may not always be initialized when they are declared. If you do not initialize a variable it takes its default value according to its data type. Table 4 lists the default value of all the data types:

Table 4: Default Values of Data Types

Data Type	Default value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d

Table 4: Default Values of Data Types

Data Type	Default value
char	'\u0000'
boolean	false

All data types have their default values until some other value is not assigned to them.

You have learned different data types and their default values available in Java. Now, let's discuss about the variables in the following section.

2.2 Declaring Variables

Java provides us containers or placeholders to store data within a program. These containers are called variables since the values stored in them can be changed during the course of execution of a program. To declare a variable in a Java program, you need to use the combination of a data type and an identifier. An identifier is the name given to the variable and it must be a logical name. A variable must be declared before it is used in the program. The syntax for declaring a variable is:

```
dataType variable;// declaring variable
```

In the preceding syntax, `dataType` specifies a valid Java data type, such as `int`, char, and so on. `variable` can be any name that follows some specified naming conventions, which are discussed later in this topic.

Apart from declaring a single variable in a declaration statement, you can also declare more than one variable of a particular data type. The syntax for declaring more than one variable in a declaration statement is given here:

```
dataType variable1, variable2,....variableN;// declaring more than one
 variable
```

As shown in the preceding syntax, if you declare more than one variable at a time, they must be seperated by comma (,). You must always remember to terminate a variable declaration by semicolon (;). After declaring the variable, let's now see how to initialize it in the following syntax:

```
dataType variable = val;// initializing a variable
```

In the preceding syntax, `val` represents the value that is used to initialize the variable. Apart from initializing a variable at the time of its declaration, it can also be initialized later after its declaration. The following syntax shows how a variable is initialized after its declaration:

```
dataType variable;
.
.
variable=val;
```

In the preceding syntax, the variable is declared first and initialized later.

Variables in Java are categorized into the following three main types:

- Instance Variables
- Class Variables
- Local Variables

Let's discuss all these variables in detail in the following sections.

Declaring the Instance Variables

Instance variables are declared inside a class as its member and created when the objects of the class are instantiated. The instance variables are associated with objects and can take different values for each object. The instance variable allocates separate memory space for each object. Instance variables are available in the program till their associated objects are available. Once the associated object goes out of scope, the associated instance variable is no longer accessible. The following code snippet declares the instance variables:

```
class InstanceVar {
  int var1;
  .
  .
  public static void main(String ar[]) {
      InstanceVar obj1=new InstanceVar();
      InstanceVar obj2=new InstanceVar();
  }
}
```

In the preceding code snippet, the `var1` variable is an instance variable. When the `obj1` and `obj2` objects are instantiated, seperate memory space is allocated to `var1` for each object.

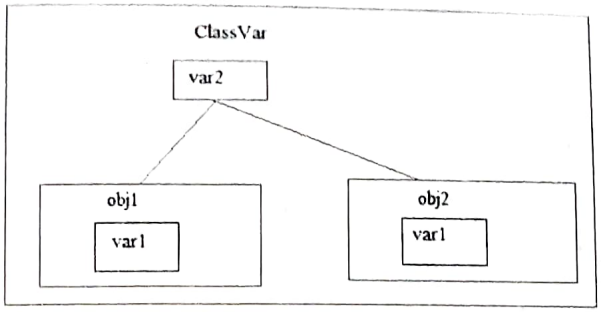
Declaring the Class Variables

Class variables are also declared inside the class as its member, but with the `static` keyword. Class variables are associated with that class and they allocate only one memory space for all the objects of that class. In other words, the same value of the class variable is shared by all the objects of that class. Class variables are available in the program till the existence of their associated class. The following code snippet declares the class variables:

```
class ClassVar {
  int var1;
  static int var2;
  .
  .
  public static void main(String ar[]) {
      ClassVar obj1=new ClassVar();
      ClassVar obj2=new ClassVar();
  }
}
```

In the preceding code snippet, `var1` is the instance variable, whereas `var2` is the class variable.

Figure 1 shows the difference between instance variable and class variable:



▲ **Figure 1: Difference between Instance and Class Variables**

In Figure 1, the `var1` variable allocates separate memory spaces for `obj1` and `obj2`, whereas the `var2` variable allocates single memory space for both the objects. The `ob1` and `obj2` objects have their separate copy of the `var1` instance variable whereas both the objects share the same copy of the `var2` class variable.

Declaring the Local Variables

Apart from declaring instance variables and class variables, you can also declare local variables in a class. Local variables are declared inside a block or a method. The local variable declared in a block or method is only visible to that block or method. It cannot be accessed outside the block or method. Local variable are alive till the execution of their respective block of declaration. The following code snippet declares the local variables inside a method:

```
class LocalVar {
  int var1;
  static int var2;
  .
  .
  void method1() {
      int lclvar=10;
      .
      .
  }
}
```

In the preceding code snippet, the `lclvar` variable is declared inside the `method1()` method. The `lclvar` variable can only be used inside the `method1()` method. Similarly, local variables can also be declared inside a block defined between a opening brace, {, and closing brace, }. The variables declared inside a block are visible only inside that program block. The following code snippet declares the local variables inside a program block:

```
class LocalVar {
  int var1;
  static int var2;
  .
  .
  {
      int localvar=10;
      .
      .
  }
}
```

In the preceding code snippet, the `localvar` variable is a local variable declared inside a program block. The `localvar` variable is not accessible outside the block where it is declared. Apart from having a single block, you can have nested blocks also in your program. These nested blocks can have their local variable declarations. You must not declare a local variable in inner block with the name same as the variable name declared in the outer block, in case of nested blocks. The following code snippet shows how variables are declared in nested blocks:

```
class Localvar {
  int var1;
  static int var2;
  .
  .
  {
      int localvar=10;
      .
      .
      {
          char grade='A';
          .
          .
      }
      {
          int roll=1;
          .
          .
      }
  }
}
```

In the preceding code snippet, the `localvar`, `grade`, and `roll` local variables are declared in nested program blocks. The variables (`localvar`) declared in outer block are accessible inside the inner block, whereas the variables inside the inner blocks (`grade`, `roll`) are not accessible inside the outer block.

.3 Declaring Classes

Java is pure object-oriented programming language, which implies that all the instructions in a Java program must be written inside the class. Class is a template to create Java programs consisting of attributes and methods. Attributes are the data member of the class, which are manipulated and accessed by the methods of the class. Unlike `C` and `C++`, methods in normal Java classes are defined at the time of its declaration. The `class` keyword is used to declare a class. Following syntax declares a class in Java:

```
class <class_name> // Class declaration
{
// declaring variables
  datatype <var_name1>;
  datatype <var_name2>;
// defining methods
  return_type method_name (parameters)
  {
      //Method body
```

```
    }
}
```

In the preceding code snippet, only two variables are declared but you can create as many as required. In the same way, number of methods that need to be used in a class depends on the requirement of the program. A Java class, which is the starting class of your program must contain the `main()` method as you need to call the class containing `main()` method, to execute the Java program.

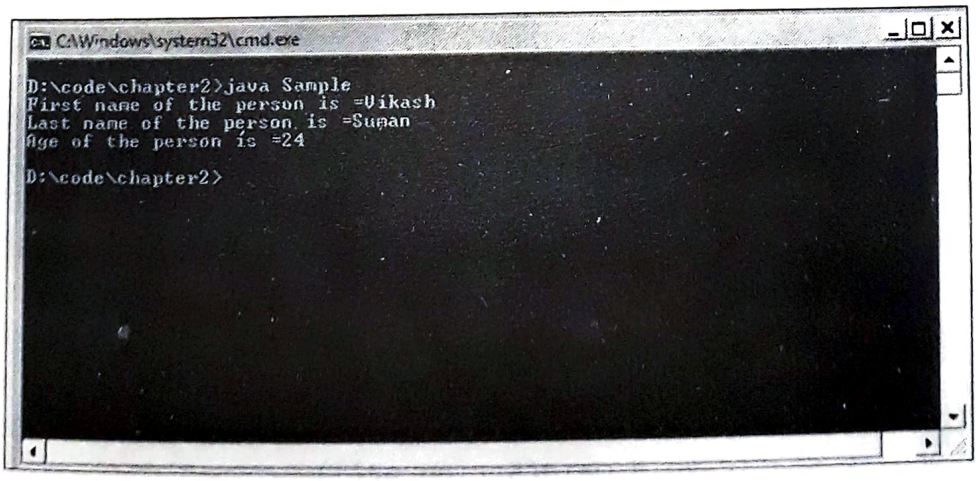
Applets do not need main methods. Applets are Java programs that can be included in Web pages.

Listing 1 shows an example to understand how to create a class (you can find the `Sample.java` file on the CD in the code\chapter2 folder):

▶ **Listing 1: Creating a Sample Java Class**

```
class Sample {
  public static void main (String args[])  {
      // variable declaration
      String fname;
      String sname;
      int age;
      // Assigning values to the variables declared
      fname = "Vikash";
      sname = "Suman";
      age = 24;
      //Displaying the values assigned
      System.out.println ("First name of the person is =" +  fname);
      System.out.println ("Last name of the person is =" +  sname);
      System.out.println ("Age of the person is =" + age);
  }
}
```

In Listing 1, the `Sample` class is created. This class contains the `main()` method, which is the very first method called during the execution of a Java program. In Listing 1, first we have declared some variables such as `fname`, `sname`, and `age`, which are then followed by some values assigned to those variables. At last, we have displayed the value assigned to those variables. Figure 2 shows the output of the Listing 1:



▲ **Figure 2: Output of the Sample Program**

In Figure 2, you can see that the values assigned to the variables are displayed along with the statements specified within the `System.out.println()` statement. Apart from simple classes, you can also create classes within classes in Java. Let's discuss about nested classes in the following section.

Declaring Nested Classes

Java allows you to create classes within classes. A class declared within another class is called nested class. A nested class has following features:

- The scope of the nested classes is within the class where it is declared. In other words, if a class Address is declared within the class Employee, then the Address class is known to the Employee class only and not outside the Employee class.
- A nested class can access members of the enclosing class including its private members, whereas the enclosing class does not have access to members of the nested class.

In Java, nested classes are of two types mentioned as follows:

- **Static nested classes:** A nested class declared with the `static` keyword or within a static context of enclosing class is called static nested class. A static nested class cannot access the instance variables of the class directly; rather it needs the object of enclosing class to access its member. Due to this limitation, the static nested class is not preferably used.
- **Non-static nested classes:** A normal nested class declared without using the static keyword is called non-static nested class. It is also called inner class. The non-static nested classes can access the instance variables of the enclosing classes directly, without instantiating it. Therefore, you can conclude that the non-static nested class is fully within the scope of the enclosing class.

Listing 2 shows an example of nested class (you can find the NestedDemo.java file on the CD in the code\chapter2 folder):

▶ ***Listing 2: Creating a Nested Java Class***

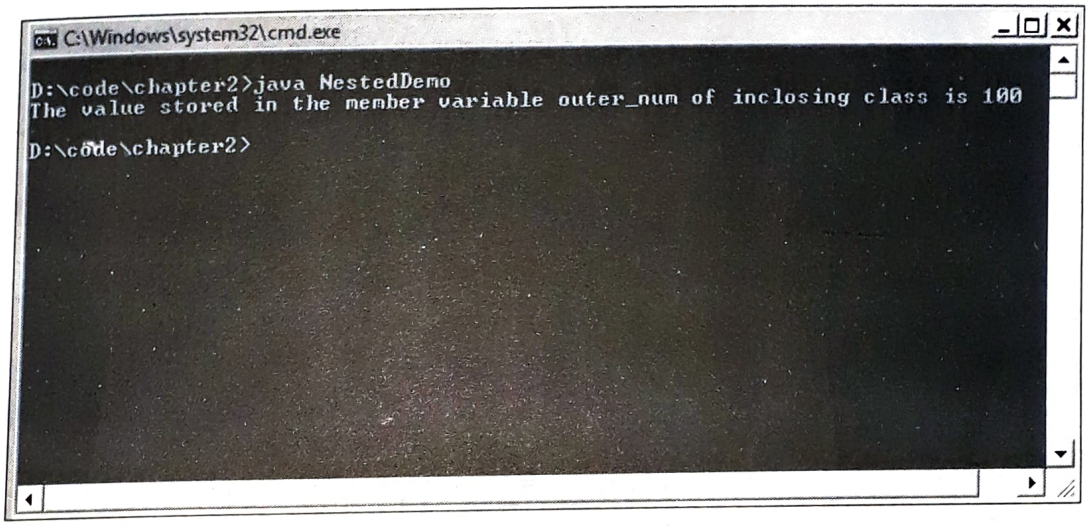
```
class Enclosing
{
 int outer_num=100;
 void display()
 {
     Nested n=new Nested();
     n.showData();
 }

 class Nested
 {
     void showData()
 {
 System.out.println("The value stored in the member variable outer_num of
 inclosing class is "+outer_num);
 }
 }
}

class NestedDemo
{
 public static void main(String ar[])
 {
```

```
        Enclosing obj=new Enclosing();
        obj.display();
    }
}
```

In Listing 2, the `Enclosing` class is the outer class and the `Nested` class is the inner class. The inner class accesses the instance variable of the outer class directly and displays its value. Inside the main() method, the obj object of the `Enclosing` class is created, which invokes the `display()` method. The `display()` method in turn invokes the `showData()` method of the `Nested` class and displays 100 as output. Figure 3 shows output of the program given in Listing 2:



▲ **Figure 3: Output of the NestedDemo Program**

2.4 Declaring Methods

A Java class consists of attributes and methods. Methods are considered as functions that are used to perform some specific task in an application. A method can have any name, such as `disp`, `vol`, and so on. Depending on the requirement of the program, many methods can be created in a program. The syntax to create a method is as follows:

```
<return_type> <method_name> ([para_list])
{
  // method body;
}
```

In the preceding syntax, `return_type` can be any data type depending on the type of values that a method is returning. If the method is not returning any value, then the data type is replaced with void. A method returns the value to calling routine through the return statement. The `method_name` is the name of the method. The `para_list` represents a list of parameters that are passed in a method. It may happen that there is no need to use parameters in a method. In that case, the `para_list` is not used and brackets remain empty. Within the curly braces, the body of the method is written.

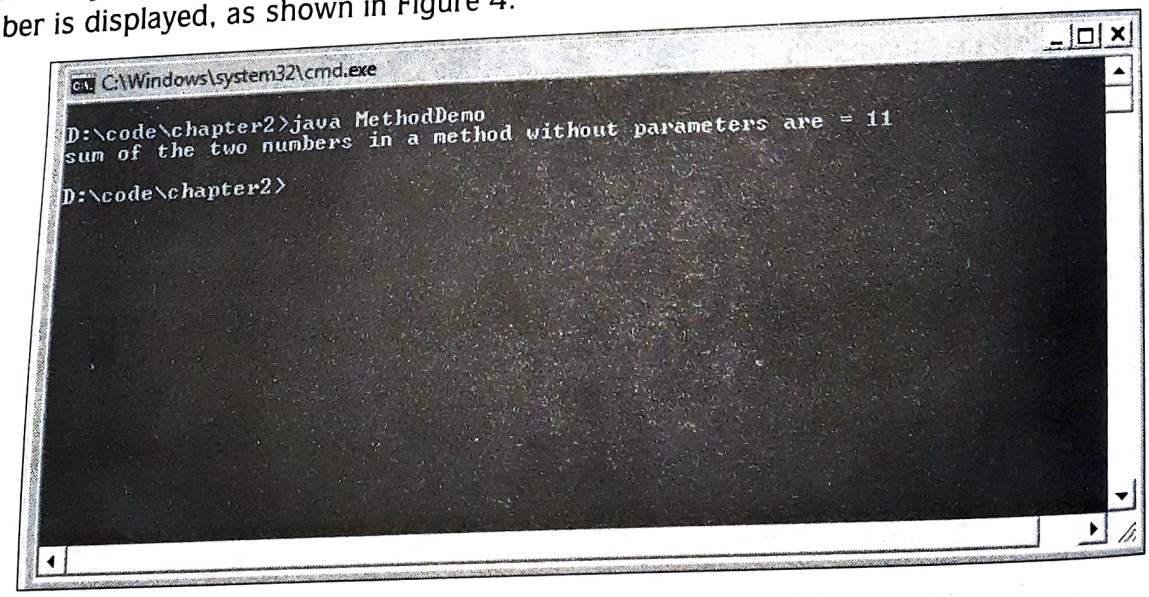
Listing 3 shows how to create and use methods (you can find the `MethodDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 3: Creating and Using Methods**

```
class MethodDemo
{
  void no_para()
  {
      int j=5;
      int k=6;
      int s= j+k;
      System.out.println("sum of the two numbers in a method without
      parameters are = " + s);
  }
  public static void main(String args[])
  {
      MethodDemo get_sum = new MethodDemo();
      get_sum.no_para(); //  Invoking  the no_para method
  }
}
```

In Listing 3, the `MethodDemo` class is created with the `no_para()` method. The `no_para()` method is used to calculate the sum of two numbers. In the `no_para()` method, two variables are declared and initialized with some value. After that, we have used '+' operator to calculate their sum, which is then followed by assigning the sum into a third variable. Finally, the sum assigned in the third variable is displayed along with the statement written within '`System.out.println()` statement'.

Inside the `main()` method, the `get_sum` instance of the `MethodDemo` class is created. Then this object is used to invoke the `no_para()` method. The `no_para()` method is invoked with no parameter values because it is a non-parameterized method. (The method defined without any parameter is known as non-parameterized method). As a result, the sum of the number is displayed, as shown in Figure 4:



▲ **Figure 4: Output of the MethodDemo Program**

In Figure 4, you can see the output of Listing 3. This is how methods are created and used. The `no_para()` method in the `MethodDemo` class does not return any value.

TEST YOUR KNOWLEDGE

Q1. Write a program to add two numbers with the use of method.

Ans.

```
class TestYourKnowledge1 {
 public void sum(int a, int b) {
     system.out.println(a+b);
 }
 public static void main(String ar[]) {
     TestYourKnowledge1 ob=new TestYourKnowledge1 ();
     int a=10, b=20;
     ob.sum(a,b);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge1.java
D:\code\chapter2>java TestYourKnowledge1
```

Output:

```
30
```

Although, you can create methods, which return some value also. While using these methods, you can pass either a value or an object reference as method parameter. Apart from these activities with methods, you can overload and override the methods in your program. Let's discuss all these activities performed with methods in the following sections.

Methods Returning a Value

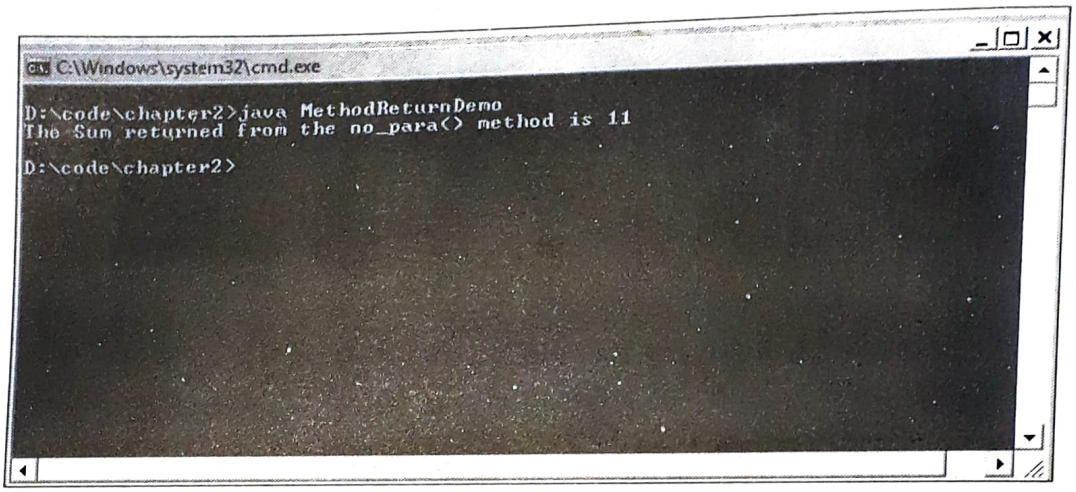
Apart from creating methods, which does not return any value to the caller, you can also create methods returning some value to the caller. For example, instead of printing the value of sum of the numbers provided in the `no_para()` method, you can return the sum to the caller. Listing 4 shows how methods returning values are created and used (you can find the `MethodReturnDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 4: Creating and Using Method Returning a Value**

```
class MethodReturnDemo
{
 int no_para()
 {
     int j=5;
     int k=6;
     int s= j+k;
     return s;
 }
 public static void main(String args[])
 {
     MethodReturnDemo get_sum = new MethodReturnDemo();
     int sum=get_sum.no_para();
     System.out.println("The Sum returned from the no_para() method is "+
     sum);
```

```
    }
}
```

In Listing 4, the `MethodReturnDemo` class is created with the `no_para()` method. The `no_para()` method calculates the sum of two numbers and returns the sum to the caller block. As this method is called from the `main()` method, the sum is returned in the `main()` method. Finally, the sum is displayed inside the `main()` method. Figure 5 shows the output of Listing 4:



▲ **Figure 5: Output of the MethodReturnDemo Program**

As shown in the Figure 5, the sum of 5 and 6 is displayed to the standard output. The `no_para()` method in the `MethodReturnDemo` class has no parameter. You can also pass parameters to the methods.

TEST YOUR KNOWLEDGE

Q2. Write a program to demonstrate the use of a method, which returns a value.

Ans.

```
class TestYourKnowledge2 {
 public int square(int a)  {
     return a*a;
 }
 public static void main(String ar[]) {
     TestYourKnowledge2 ob=new TestYourKnowledge2 ();
     int a=10;
     int sq=ob.square(a);
     System.out.println(sq);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge2.java
D:\code\chapter2>java TestYourKnowledge2
```

Output:

```
100
```

Let's discuss how methods with arguments are created and used in the following section.

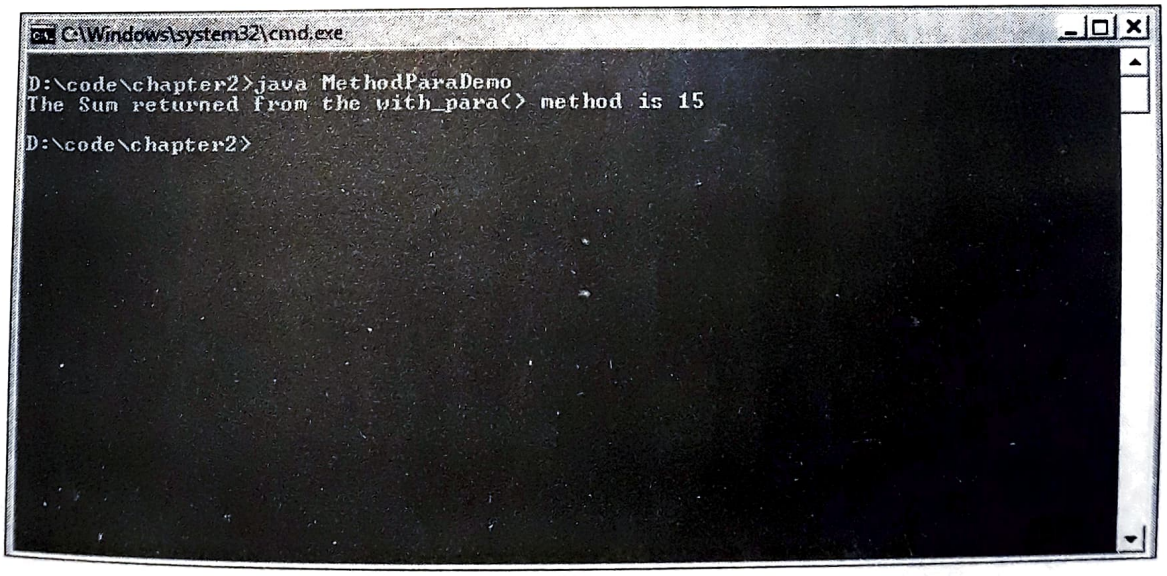
Passing Arguments to Methods

In Java, you can create methods, which accept argument. The methods with arguments are called parameterized methods. These methods can accept all the data type available in Java. The parameter is passed at both the places, when the method is invoked and when the method is defined. The parameters passed at the time of invocation of method are called actual parameter, and the parameter passed at the time of its definition is called dummy parameter. Listing 5 shows how to create and use methods with parameter (you can find the `MethodParaDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 5: Creating and Using Method with Arguments**

```
class MethodParaDemo {
 int with_para(int j, int k) {
     int s= j+k;
     return s;
 }
 public static void main(String args[])   {
     MethodParaDemo get_sum = new MethodParaDemo();
     int sum=get_sum.with_para(7,8);
     System.out.println("The Sum returned from the with_para() method is "+
     sum);
 }
}
```

In Listing 5, the `MethodParaDemo` class is created with the `with_para()` method. This method accepts two integer arguments and returns the sum of those two arguments. Sum is retrieved inside the `main()` method and displayed there. The parameters 7 and 8 are actual parameter, which is passed when the method is called. The parameters `j` and `k` are dummy parameters, which are passed when the method is defined. Figure 6 shows the output of Listing 5:



▲ **Figure 6: Output of the MethodParaDemo Program**

In Figure 6, sum of 7 and 8 is calculated inside the `with_para()` method and displayed in the main() method.

The parameters passed to the `with_para()` method inside the `main()` method are constant values, you can also pass variables containing those values. Let's discuss the effects of passing values and object references to the methods in the following sections.

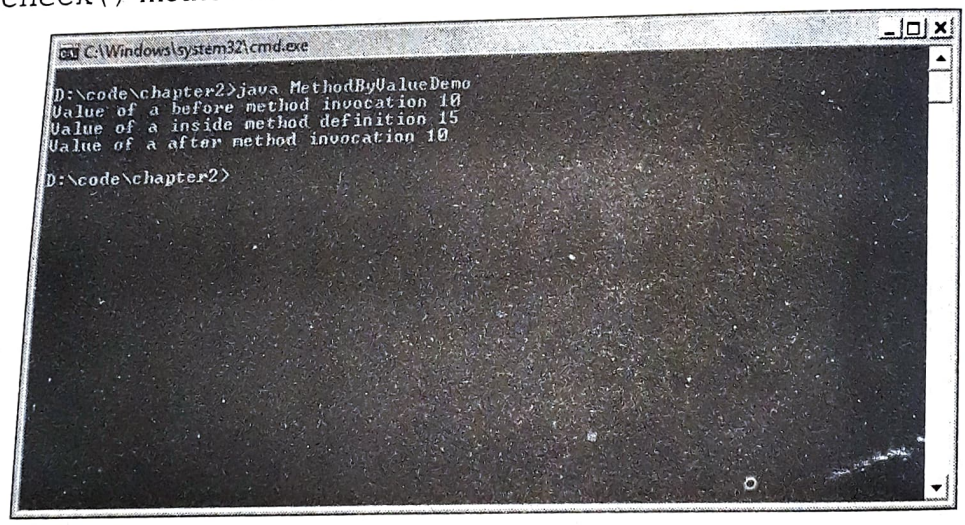
Call by Value

When values are passed to the methods at the time of their invocation, this type of method call is called `call by value`. Listing 6 shows an example of call by value (you can find `MethodByValueDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 6: Implementing Call by value**

```
class MethodByValueDemo {
  void check(int a)  {
      a=a+5;
      System.out.println("Value of a inside method definition "+a);
  }
  public static void main(String args[])   {
      MethodByValueDemo obj = new MethodByValueDemo();
      int a=10;
      System.out.println("Value of a before method invocation "+a);
      obj.check(a);
      System.out.println("Value of a after method invocation "+a);
  }
}
```

In Listing 6, the `MethodByValueDemo` class is created with the `check()` method. The `check()` method is invoked within the `main()` method. A variable containing value 10 is passed to the `check()` method. The value of `a` variable is displayed before and after the invocation of the `check()` method and it remains same. While the value of other variable is changed inside the `check()` method definition. The `System.out.println()` statement inside the `check()` method shows the changed value of `a`, as shown in Figure 7:



▲ **Figure 7: Output of the MethodByValueDemo Program**

In Figure 7, the value of `a` inside the `main()` method remain same even after the invocation of the `check()` method. Although, the value of `a` is increased by 5 inside the `check()` method definition. Therefore, you can conclude that there is no effect of modification on the actual parameters of the methods. This is because, separate memory are allocated by the actual and the dummy parameters.

Apart from passing values as parameter, references can also be passed as method parameter. Let's discuss how references can be passed as the method parameter and the effects of passing reference as parameter in the following section.

Call by Reference

When you pass the object reference to the methods as its parameter, such method invocations are called call by reference. In call by reference, any change made to the dummy parameter is also reflected in the actual parameter. This is because; both the actual and dummy parameters refer to the same object. Listing 7 shows how methods are created and called by `call by reference` (you can find `MethodByRefDemo.java` file on the CD in the code\chapter2 folder):

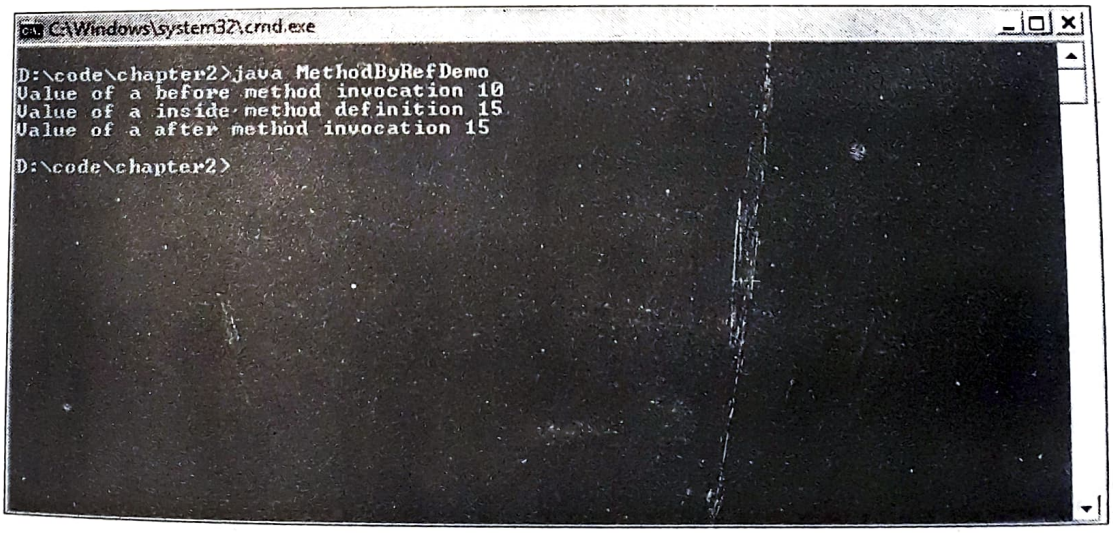
▶ Listing 7: Implementing Call by Reference

```
class MethodByRefDemo {
  int a=10;
  void check(MethodByRefDemo obj)   {
      obj.a=obj.a+5;
      System.out.println("Value of a inside method definition "+a);

  }
  public static void main(String args[])   {
      MethodByRefDemo obj = new MethodByRefDemo();
      System.out.println("Value of a before method invocation "+obj.a);
      obj.check(obj);
      System.out.println("Value of a after method invocation "+obj.a);

  }
}
```

In Listing 7, the `MethodByRefDemo` class is created with the `check()` method. The `check()` method accepts an object reference as its parameter. Inside the main() method, the value of member variable `a` is displayed before the invocation of the `check()` method. The `check()` method is invoked with the `obj` reference of the `MethodByRefDemo` class. Inside the `check()` method definition, the value of `a` member variable is increased by 5. This change is visible inside the `check()` method as well as outside the `check()` method, as shown in Figure 8:



▲ **Figure 8: Output of the MethodByRefDemo Program**

Figure 8 shows that the value of a is 10 before the invocation of the `check()` method. Inside the `check()` method definition, the incremented value is displayed. Even in the `main()` method, the incremented value of a is displayed. Therefore, you can conclude that the modification done in a method is reflected outside the method also when you invoke methods through `call by reference`.

In all the preceding listings related to method example, only one method is created and used. You can create more than one method according to your requirement. You can even create and use more than one method with same name in your program. Let's discuss how more than one method with same name can be created in a program in the following section.

Overloading and Overriding of Methods

Method overloading is an object-oriented technique that lets you define several different versions of a method, all with the same name, but each with a different parameter list. When you use overloaded methods, the Java compiler decides which method is to be invoked by the number and/or types of the parameters you pass to the method.

Methods are overloaded on the basis of the following facts:

- Number of Parameters
- Type of Parameters
- Order of Parameters

Method overriding is also an object-oriented technique that lets you define methods with the same name in the `superclass` and `subclass` when you implement inheritance. When you use method overriding, the Java compiler decides which method is to be invoked by the object, which is used to invoke that method.

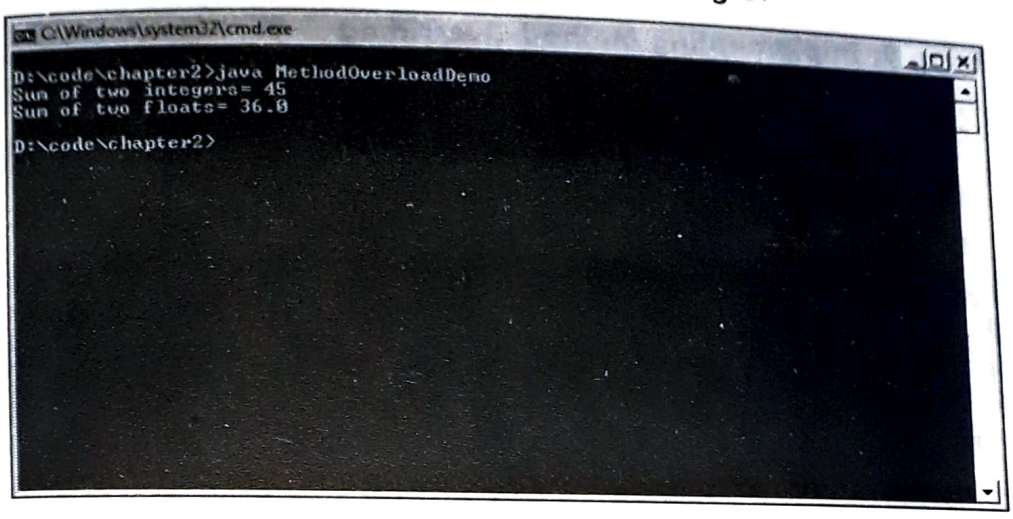
Let's take an example of method overloading. Listing 8 shows how methods are overloaded in a Java program (you can find the `MethodOverloadDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 8: Implementing Method Overloading**

```
class MethodOverloadDemo {
  void sum(int a, int b)     {
      int sum=a+b;
      System.out.println("Sum of two integers= "+sum);
  }
  void sum(float x, float y)     {
      float sum=x+y;
      System.out.println("Sum of two floats= "+sum);
  }
  public static void main(String args[])     {
      MethodOverloadDemo obj = new MethodOverloadDemo();
      obj.sum(20,25);
      obj.sum(15.5f,20.5f);
  }
}
```

In Listing 8, the `MethodOverloadDemo` class is created with two `sum()` methods. In the first `sum()` method, two integers are passed as method parameter whereas, in the second `sum()` method, two floats are passed as method parameter. Therefore, these two methods are treated as two different methods by the Java compiler. The method with integer

parameters is invoked when you pass integer parameters at the time of its invocation. On the other hand, the method with float parameters is invoked when you pass float parameters at the time of its invocation. Figure 9 shows the output of Listing 8:



▲ **Figure 9: Output of the MethodOverloadDemo Program**

In Figure 9, when 20 and 25 are passed as parameters of the `sum()` method, the `sum()` method with integer parameter is invoked and displays 45 as output. Whereas, when 15.5f and 20.5f are passed as method parameter, they invoke the `sum()` method with float parameter and display 36.0 as a result.

TEST YOUR KNOWLEDGE

Q3. Write a program to calculate the area of square and rectangle by overloading the area method.

Ans.

```
class TestYourKnowledge3 {
  public void area(int a)   {
      System.out.println("Area of Square is: "+(a*a));
  }
  public void area(int length, int width) {
      System.out.println("Area of Rectangle is: "+(length*width));
  }
  public static void main(String ar[])  {
      TestYourKnowledge3 ob=new TestYourKnowledge3 ();
      ob.area(5);
      ob.area(5,6);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge3.java
D:\code\chapter2>java TestYourKnowledge3
```

Output:

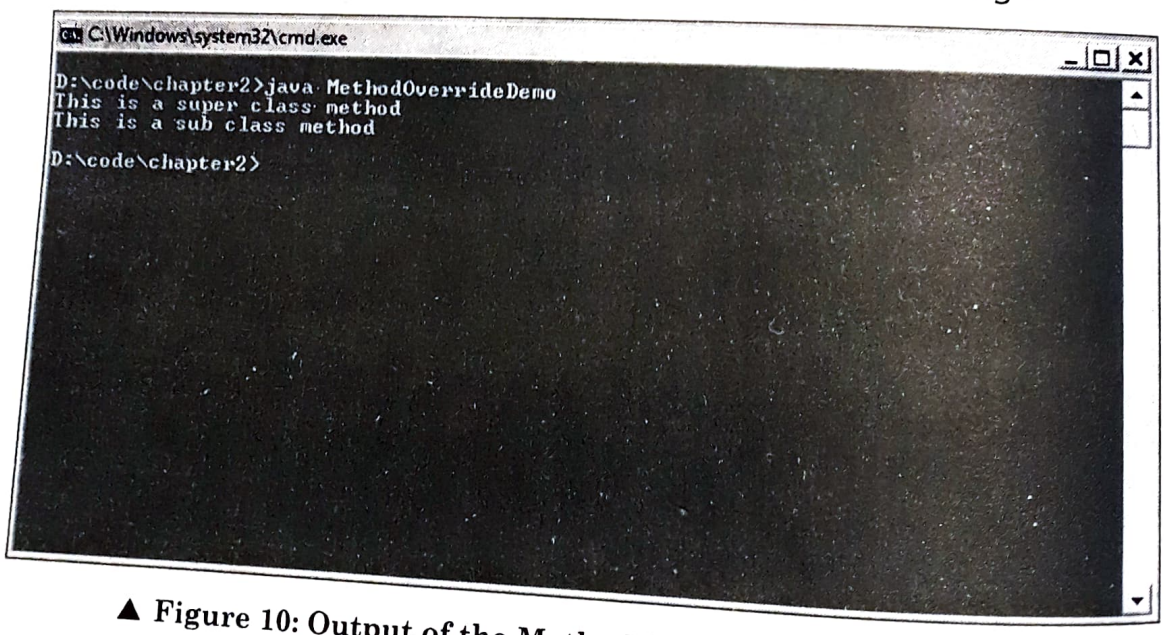
```
Area of Square is: 25
Area of Rectangle is: 30
```

After discussing all about the method overloading, let's take an example of method overriding. Listing 9 shows how methods are overridden (you can also find the `MethodOverrideDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 9: Implementing Method Overriding**

```
class SuperClass {
  void display() {
      System.out.println("This is a super class method");
  }
}
class SubClass {
  void display() {
      System.out.println("This is a sub class method");
  }
}
class MethodOverrideDemo {
  public static void main(String ar[]) {
      SuperClass objsuper=new SuperClass();
      SubClass objsub=new SubClass();
      objsuper.display();
      objsub.display();
  }
}
```

In Listing 9, three classes are created with an implementation of method overriding. The `SuperClass` class is the super class of the `SubClass` class. Both the `SuperClass` and `SubClass` classes have the `display()` method. The `main()` method is created inside the `MethodOverrideDemo` class, where both `display()` methods of the `SuperClass` class and the `SubClass` class are invoked. Figure 10 shows the output of Listing 9:



▲ **Figure 10: Output of the MethodOverrideDemo Program**

In Figure 10, when the object of the `SuperClass` class is used to invoke the `display()` method, the `display()` method of the `SuperClass` class is invoked and displays "This is a super class method" as output. Whereas, when the object of the `SubClass` class is used to invoke the `display()` method, the `display()` method of the `SubClass` class is invoked and displays "This is a sub class method" as output.

In method overriding, the method signatures must be same in the super class and sub class.

After discussing about methods, let's discuss a similar member of class i.e. constructor in the following section.

2.5 Explaining Constructors

Constructors are special type of methods having the same name as the class name, for example, if a class has the name `Demo` then the constructor will also have the name `Demo`. Constructors are syntactically similar to methods and are used to initialize the objects. A constructor initializes the object as soon as the object is created. Following are few important points to remember while working with constructors:

- The name of the constructor should be same as that of a class.
- A constructor does not have any return type—not even void—because implicitly, the return type of a constructor is class itself.
- First line of a constructor must either be a call to other constructor in the same class or a call to super class constructor. If you do not call other constructor or super class constructor explicitly, then the compiler implicitly calls the parameter less constructor of its super class.

Now, we discuss the types of constructors and how to create and use constructors under the following subheadings:

- Defining Simple Constructors
- Defining Parameterized Constructors
- Overloading Constructors

N CS889

Let's discuss these in detail.

Defining Simple Constructors

A constructor without any parameter is called a default constructor or simple constructor. A default constructor is automatically created and invoked by `JVM`. When no constructor is created explicitly then Java first implicitly creates a constructor without any parameter and invokes it. The default constructor then initializes the instance variables to default values, that is Zero for numeric data types, `null` for object reference and String type, and false for `boolean`.

Default constructors are used implicitly by Java when no other constructor is available. If you create any default constructor (without parameter) explicitly then Java uses the constructor created by you to initialize the instance variable related to objects of a class. Listing 10 shows how to create and use default constructor (you can find the `DefaultConstructorDemo.java` file on the CD in the code\chapter2 folder):

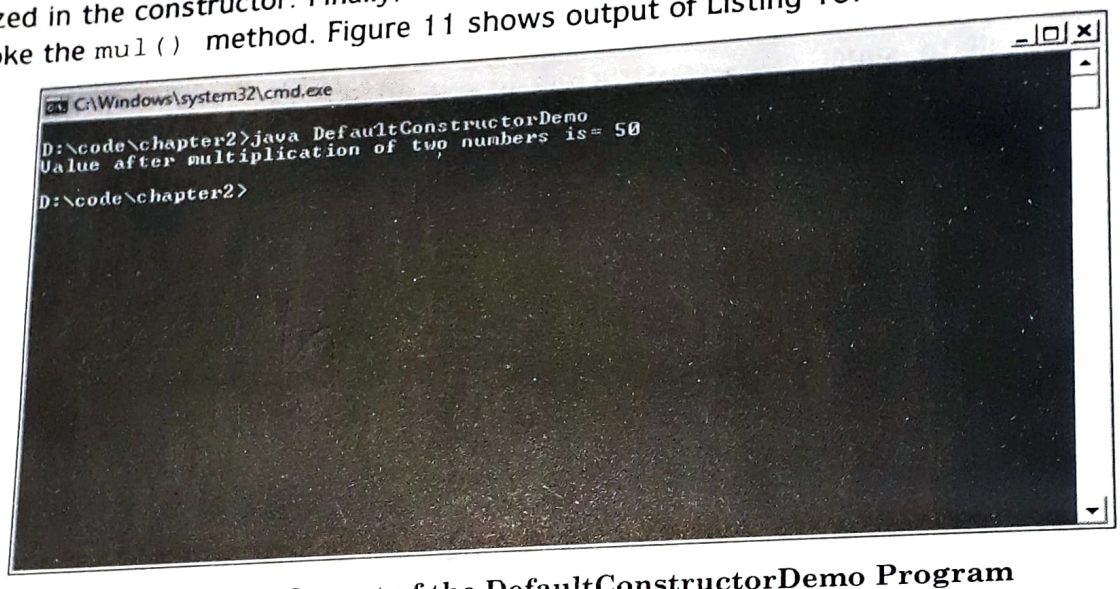
▶ Listing 10: Creating Default Constructor

```
class DefaultConstructorDemo {
  int x, y;
  DefaultConstructorDemo () // simple constructor {
      x=10;
      y=5;
```

```
    }
    public void mul()  {
        int m=x*y;
        System.out.println ("Value after multiplication of two numbers is=
        " + m);
    }
    public static void main (String args[])  {
        DefaultConstructorDemo chk = new DefaultConstructorDemo();
        chk.mul();
    }
}
```

In Listing 10, the `DefaultConstructorDemo` class is created to show how to create and use a default constructor in a class. In the `DefaultConstructorDemo` class, first two instance variables `x` & `y` are declared. Then, a constructor without parameter is created in which variables `x` & `y` are initialized with some values.

Then, the `mul()` method is created to calculate the product of the instance variable that are initialized in the constructor. Finally, we have created an object called `chk`, which is then used to invoke the `mul()` method. Figure 11 shows output of Listing 10:



▲ **Figure 11: Output of the DefaultConstructorDemo Program**

In Figure 11, when the `chk` object of the `DefaultConstructorDemo` is created, the constructor created in this class is invoked and the variables are initialized. As a result, the `mul()` method calculates the multiplication value as 50.

Defining Parameterized Constructors

The constructors having parameters are known as parameterized constructors. In parameterized constructors, the instance variable is initialized automatically at run time according to the values passed to parameters during the object creation. Parameterized constructor allows us to initialize the instance variable with different values; whereas, in case of default constructor, the values will remain same for all objects. For example, in case of default constructor, we have to initialize the instance variables with some fixed value and that value remains the same for all the objects of the class. In case of parameterized constructor, we do not initialize the instance variable with some fixed values; rather we use parameters to initialize instance variables. Parameters used in parameterized constructor receive the value

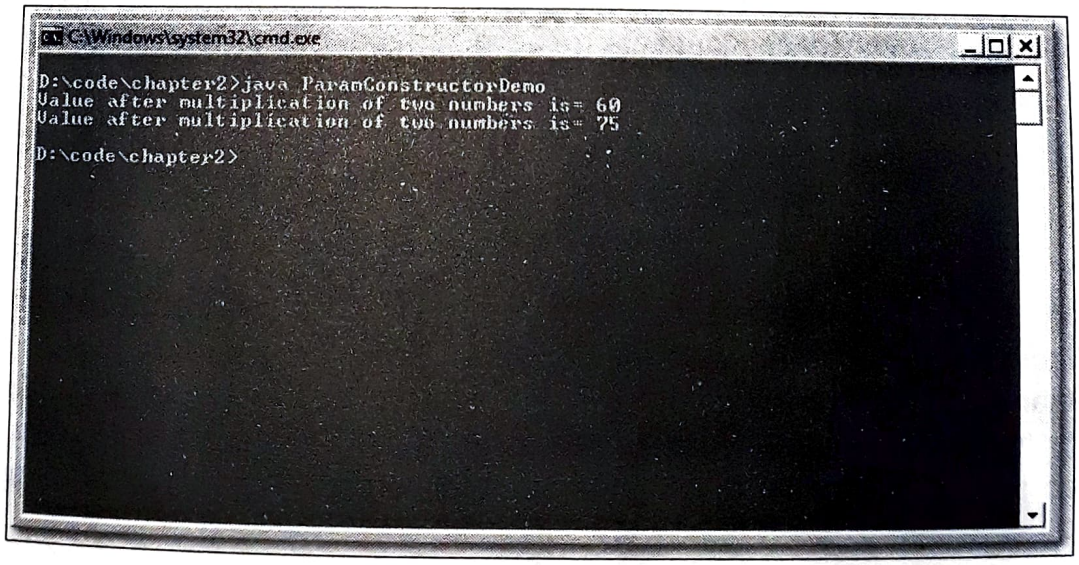
passed during object creation and assign those values to instance variables. Listing 11 shows how to create and use parameterized constructor (you can find the ParamConstructorDemo.java file in CD in the code\chapter2 folder):

▶ **Listing 11: Creating Parameterized Constructor**

```
class ParamConstructorDemo {
 int x, y;
 ParamConstructorDemo (int a, int b) // parameterized constructor {
     x=a;
     y=b;
 }
 public void mul () //method to calculte the product of two numbers {
     int m=x*y;
     System.out.println ("Value after multiplication of two numbers is= " +
     m);
 }
 public static void main (String args[])  {
     ParamConstructorDemo chk = new ParamConstructorDemo(12, 5);
     ParamConstructorDemo chk1 = new ParamConstructorDemo(15, 5);
     //invoking mul method
     chk.mul();
     chk1.mul();
 }
}
```

In Listing 11, the ParamConstructorDemo class is created to show how to create and use a parameterized constructor in a class. In the ParamConstructorDemo class, two instance variables, x & y are declared. Then a parameterized constructor is created in which the variable x & y are initialized with some dynamic values through the parameters used in constructor.

Then the mul()method is created to calculate the product of the instance variable that are initialized in the constructor. Finally, the chk and chk1 objects of the ParamConstructorDemo are created, which are then used to invoke the mul() method. Different values are passed while creating these objects to show that separate values will be assigned to instance variables at run time. Figure 12 shows output of Listing 11:



▲ **Figure 12: Output of the ParamConstructorDemo Program**

In Figure 12, the `mul()` method find the products of the values of instance variables and displays the result as 60 and 75.

Similar to methods, the constructors can also be overloaded. Let's discuss how constructors are overloaded in the following section.

Overloading Constructors

Similar to methods, you can also overload constructors. In other words, you can create and use more than one constructor in a class. A constructor can be overloaded on the basis of following facts:

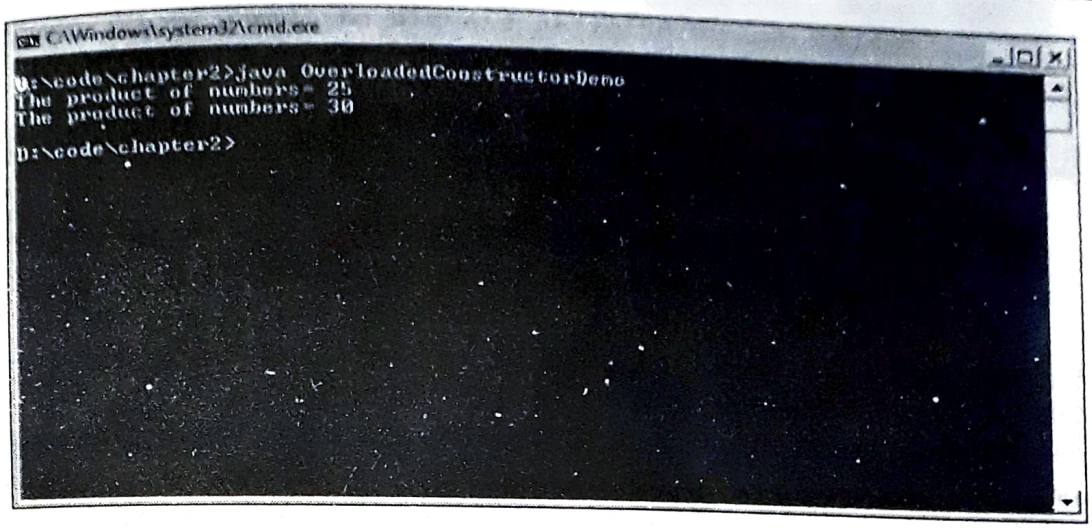
- Number of parameters
- Data type of Parameters
- Order of Parameters

When you use an overloaded constructor in your Java class, the objects of that class are created and the instance variables are initialized according to the constructor signature. Listing 12 shows how to overload constructors in your program (you can find the `OverloadedConstructorDemo.java` file on the CD in the code\chapter2 folder):

▶ Listing 12: Creating Overloaded Constructor

```
class OverloadedConstructorDemo {
 int num1,num2;
 OverloadedConstructorDemo(int a)  {
     num1=num2=a;
 }
 OverloadedConstructorDemo(int a, int b)  {
     num1=a;
     num2=b;
 }
 void calculate()   {
     int result=num1*num2;
     System.out.println("The product of numbers= "+result);
 }
 public static void main(String ar[]) {
     OverloadedConstructorDemo obj1=new OverloadedConstructorDemo(5);
     OverloadedConstructorDemo obj2=new OverloadedConstructorDemo(5,6);
     obj1.calculate();
     obj2.calculate();
 }
}
```

In Listing 12, the `OverloadedConstructorDemo` class is created with overloaded constructors OverloadedConstructorDemo(int a) and OverloadedConstructorDemo(int a, int b). When the object of the `OverloadedConstructorDemo` class is created with a single parameter, the first constructor with a single parameter is invoked. Whereas, when the object of the `OverloadedConstructorDemo` class is created with two parameters, then the second constructor with two parameters is invoked. Figure 13 shows output of Listing 12:



▲ **Figure 13: Output of the OverloadedConstructorDemo Program**

In Figure 13, depending upon the constructor invocation, the instance variables are initialized, which results in different output.

Constructors are used to create objects of a class. Let's discuss how objects are created and used in the following section.

2.6 Creating Objects

Instances of a class are known as objects and the process of creating objects from a class is known as instantiation. While creating an instance of a class, the class works as a data type. To create an object, you have to declare a variable of class type to store the object reference. For example, if you have to create an object of the Sample class (Listing 1), then you first have to declare a variable of the Sample class type. You can create the variable of the sample class as follows:

```
Sample info;
```

Now, you can create the object of the class. To create an object, the `new` operator is used that dynamically allocates memory to an object and returns a reference to it. This reference is then stored in the variable declared. Let's see how to create an object of the Sample class (Listing 1):

```
info = new Sample();
```

You can also combine both the steps to create an object, as follows:

```
Sample info = new Sample();
```

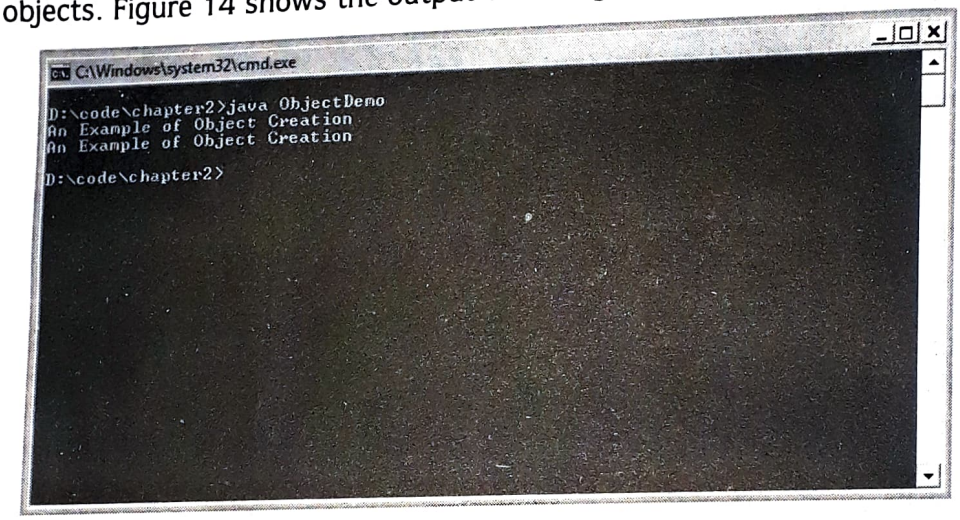
Each object has its own copy of the instance variables declared in the class definition. Listing 13 shows how the objects are created (you can find the `ObjectDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 13: Creating Objects Example**

```
class ObjectDemo {
void show()  {
```

```
        System.out.println("An Example of Object Creation");
    }
    public static void main(String ar[]) {
        ObjectDemo obj1,obj2;
        obj1=new ObjectDemo();
        obj2=new ObjectDemo();
        obj1.show();
        obj2.show();
    }
}
```

In Listing 13, the `ObjectDemo` class is created with the `show()` method. Inside the `main()` method, the objects of the `ObjectDemo` class are created. First two reference variables `obj1` and `obj2` are created and then using the `new` keyword, the objects are instantiated and assigned to these reference variables. Finally, the `show()` method is invoked using the `obj1` and `obj2` objects. Figure 14 shows the output of Listing 13:



▲ **Figure 14: Output of the ObjectDemo Program**

In Figure 14, the `show()` method is invoked two times and the message is displayed twice. The members of a class, such as method and fields and even the class itself can have a level of protection, so that they cannot be misused. Let's discuss the how the class members and the class can be protected in the following section.

.7 Explaining Access Specifiers

The members of a class have its visibility in the same class or other classes. Even the classes have their associated accessibility. How and where a member of a class is accessed is determined by access specifiers. In Java programming language, `public`, `protected`, `private`, and `default` are the available access specifiers. The access specifiers available in Java are described as follows:

- `public`: When a class member is declared with the public access specifiers, it can be accessed from anywhere in the program. You can access public classes, methods, and fields from everywhere. The public classes, methods, or fields are used only if you explicitly want to offer access to these members and if this access cannot do any harm.
- `protected`: You can access protected methods and fields from within the same class to which the methods and fields belong, within its subclasses, and within classes of the

same package, but not from other packages. The protected access specifier cannot be used with class declaration.

- `private:` You can access the private methods and fields only within the same class to which the methods and fields belong. When you declare a method or field as private, it is not visible within subclasses and is not inherited by subclasses. The private access specifier works opposite to the public access specifier. The concept of encapsulation, in which, data are hidden within the class and accessor methods are provided, is implemented through the use of private access specifier.
- `default:` If you do not specify any access specifier while declaring class, method, or field, then they are considered with default access specifier. The class, method or field with default access specifier can be accessed from within the same package to which the class, method or field belongs. These are not accessible outside the package they belong to.

You can use different types of data in your program and to store them you use different data types. Sometimes, you need to assign the content of variables of different data types. Let's discuss how it is achieved in the following section.

2.8 Explaining Type Casting and Type Conversion

While working with different data types in a program, you often need to assign value of one type of variable to another type. If the variables being used are compatible, the Java compiler performs automatic type casting. For example, you are assigning an `int` value into a long variable or a `float` value into a `double` variable, the automatic type casting takes place. Whereas, when you try to assign a `long` value into `int`, it causes compilation error. To avoid the compilation error, you need to type cast the data explicitly. Let's discuss these two types of conversions in detail in the following section.

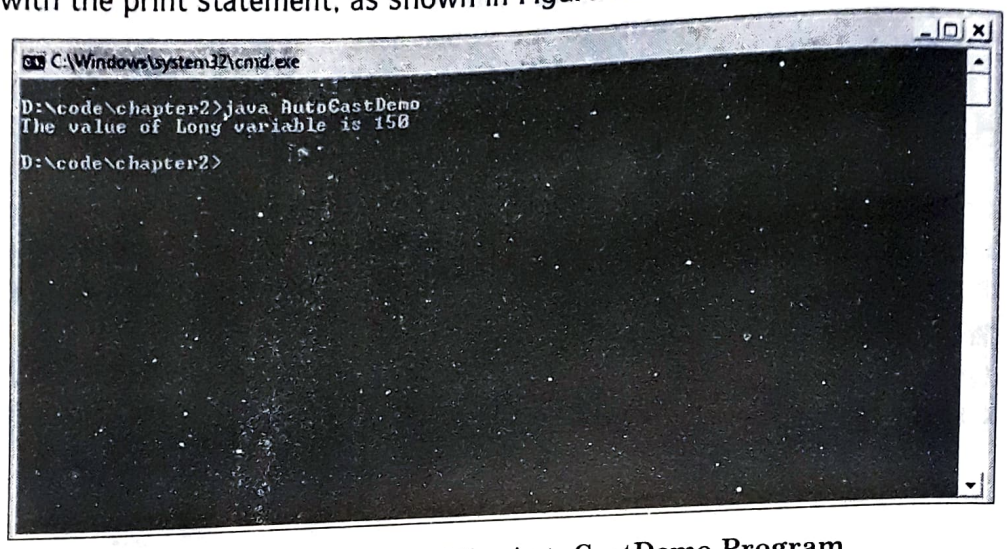
Automatic Type Casting

In the automatic type casting, it is necessary that the two data types being used must be compatible to each other. In Java, the numeric data types, such as integer and float are compatible to each other. These data types are not compatible with `boolean` and char data types. For automatic type casting, another necessary fact is that the destination data type must be longer than the source data type. When these two conditions meet then only the automatic type conversion takes place. Listing 14 shows an example of automatic type casting (you can find the `AutoCastDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 14: Automatic Type Casting Example**

```
class AutoCastDemo {
  public static void main(String ar[]) {
      int i=150;
      long l;
      l=i;
      System.out.println("The value of Long variable is "+ l);
  }
}
```

In Listing 14, the `AutoCastDemo` class is created to demonstrate automatic type casting. In the AutoCastDemo class, inside the main() method, two variables `i`, and `l` are created. The `i` variable is of integer type where is `l` is of long type. The `i` variable is initialized with value 150, which is further assigned to the `l` variable. Finally, the value of the `l` variable is displayed with the print statement, as shown in Figure 15:



▲ **Figure 15: Output of the AutoCastDemo Program**

In Figure 15, the value of the `l` variable is displayed, which was initially assigned to an `int` variable `i`.

If you want to assign the data types, which are not compatible, you need to do explicit type casting. Let's discuss how incompatible data types are cast in the following section.

Casting Incompatible Data Types

When you need to do assignment of incompatible data types, you need to write some code for explicit type casting. For example, the byte data type cannot hold `int` data type value. Following is the general syntax of the explicit type casting:

```
Target_Variable=(target_data_type) Source_Variable;
```

In the preceding syntax, the `Target_Variable` is the variable in which the data is to be assigned. The `target_data_type` is the data type of the `Target_Variable` and `Source_Variable` is the variables whose value is being assigned to the `Target_Variable`. Listing 15 shows an example of explicit type casting (you can find the `ExplicitCastDemo.java` file on the CD in the code\chapter2 folder):

▶ ***Listing 15: Explicit Type Casting Example***

```
class ExplicitCastDemo {
 public static void main(String ar[]) {
     int i;
     long l=150;
     i=(int)l;
     System.out.println("The value of Int variable is "+ i);
 }
}
```

In Listing 15, the `ExplicitCastDemo` class is created to demonstrate explicit type casting. In the `ExplicitCastDemo` class, inside the main() method two variables `i` and `l` are created. Variable `i` is of integer type whereas `l` is of long type. The variable `l` is initialized with value 150, which is further assigned to the `i` variable. Here, the long value is type cast into `int` by `i=(int)l` statement. Finally, the value of the `i` variable is displayed with the print statement, as shown in Figure 16:

▲ **Figure 16: Output of the ExplicitCastDemo Program**

In Figure 16, the value of the `i` variable is displayed, which was initially assigned to the long variable `l`. Similar to long to `int`, conversion of other incompatible data types can also be performed.

While writing programs in Java, sometimes you may need to change the flow of execution according to your program requirement. Let's discuss how the normal flow of execution of statements in a program can be altered in the following section.

2.9 Implementing Flow Control Statements

In a Java program, to achieve certain results, some specific operations need to be performed repeatedly. The execution of these operations is based upon the fulfillment of certain conditions. These conditions are placed in various control flow statements, such as conditional statements, iteration statements, and jump statements. The operation to be performed is provided in the block of control flow statements. A conditional statement is a control statement that allows choosing between two or more execution paths in a program. Iteration statements repeatedly execute the code block on the basis of the conditional statement. Once the conditional statement evaluates to false, the loop exits. After that, the control transfers to the next statement that follows the loop. Jump statements explicitly redirect the flow of program execution.

Let's discuss these flow control statements in detail in the following sections.

Using Selection Statements

Selection statements, as the name itself suggests, cause a program to execute a block of code on evaluating a conditional expression and selecting code for execution. If the conditional expression comes out to be true, the accompanying block of code is executed and if it is false, the accompanying block of code is bypassed. Following are the selection statements available in Java:

- `if` statement
- `if-else` statement
- `switch` statement

Let's discuss these selection statements in detail.

Using if Statement

The `If` statement is a conditional construct that is used to evaluate an expression for a `boolean` value. In case, the expression returns true, the statements following the conditional expression will be executed but in case, it returns false, the code following the conditional expression would not be executed and the program would just move along to the next statement. Following is the general syntax for the `if` statement:

```
if (conditional expression)
  <Statements to be executed when conditional expression evaluated to true>;
```

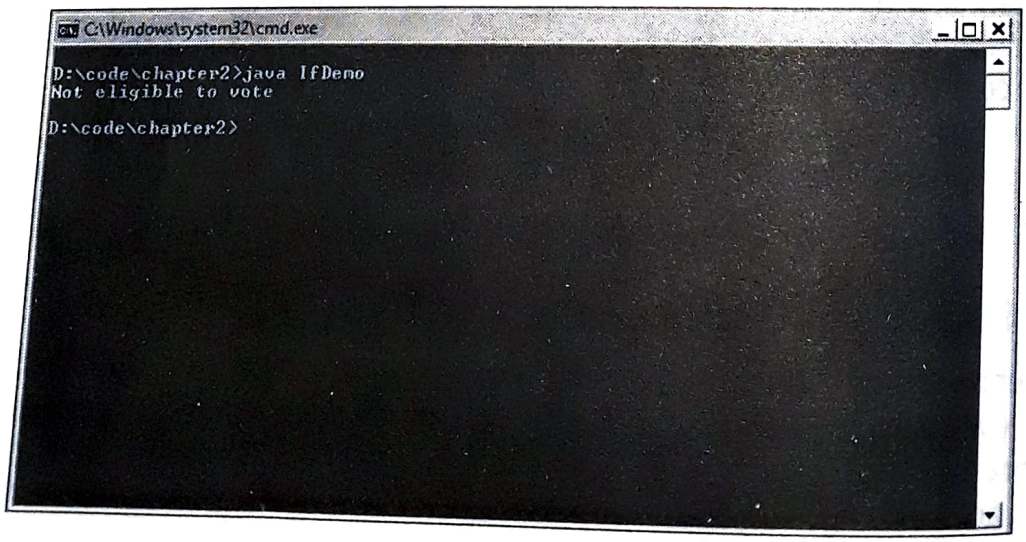
In the preceding syntax, `if` is the keyword and inside the parenthesis, the conditional expression is provided.

Listing 16 shows how to use the `if` statement (you can find the `IfDemo.java` file on the CD in the code\chapter2\ folder):

▶ ***Listing 16: Using `if` Statement***

```
class IfDemo {
 public static void main(String args[]) {
     int age;
     age=16;
     if (age<18)
     System.out.println("Not eligible to vote");
 }
}
```

In Listing 16, `if` statement is used to test a condition that is whether age specified in the program is less than 18 or not. The value for `age` variable is set to 16 in the program. Since, 16 is less than 18, the condition set in the `if` statement becomes automatically true. Therefore, the statement following the if condition is displayed, as shown in Figure 17:



▲ **Figure 17: Output of the IfDemo Program**

In Figure 17, the statement provided inside the `if` block is displayed, because the condition is `true`. Apart from executing statements when the statement is `true`, you can also write code to execute statements when the condition is `false`. Let's discuss how it can be achieved in the following section.

Using if-else Statement

The `if-else` statement is used in situations where we can provide an alternative course of action, in case, the `if` clause returns false. For example, when the conditional expression for the `if` clause evaluates to false, the statements coded in else clause gets executed.

Following is the general syntax for 'if-else' statement:

```
if (conditional expression)
  <statement1 to be executed in case it's true>;
  else
      <statement2 to be executed in case it's false>;
```

In the preceding syntax, the condition is written inside the parenthesis (). According to the preceding syntax, if the condition specified after '`if`' is true, then statement1 is executed; else, statement2 is executed. Listing 17 shows how to use if-else statement (you can find the IfElseDemo.java file on the CD in the code\chapter2 folder):

▶ **Listing 17: Using if-else Statement**

```
class IfElseDemo {
  public static void main(String[] args) {
      int a = 10, b = 20;
      if (a > b) {
          System.out.println("a > b");
      } else {
          System.out.println("b > a");
      }
  }
}
```

In Listing 17, the '`if`' statement is used to test a condition that whether value stored in variable '`a`' is greater than the value stored in variable '`b`' or not. Since the value stored in variable '`a`' is set to 10 and that of '`b`' to 20, the condition automatically evaluates to false and the statement immediately following the condition does not get executed. Then, the statement after the else construct gets executed, as shown in Figure 18:

▲ **Figure 18: Output of the IfElseDemo Program**

In Figure 18, the statement inside the else block is executed as the condition in if statement is false. In both the preceding examples, only one if statement is used, you can use multiple if in your program also. Let's discuss it in the following section.

TEST YOUR KNOWLEDGE

Q4. Write a program to display whether a person is eligible for vote or not.

Ans.

```
class TestYourKnowledge4
{
 public static void main(String ar[])
 {
     int age=18;
     if(age>=18)
     {
         System.out.println("Person is eligible for vote");
     }
     else
     System.out.println("Person is not eligible for vote");
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge4.java
D:\code\chapter2>java TestYourKnowledge4
```

Output:

```
Person is eligible for vote
```

Q5. Write a program to check whether the given number is even or odd.

Ans.

```
class TestYourKnowledge5
{
 public static void main(String ar[])
 {
     int num=5;
     if(num%2==0) {
         System.out.println("Number is even");
     }
     else
     System.out.println("Number is odd");
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge5.java
D:\code\chapter2>java TestYourKnowledge5
```

Output:

```
Number is odd
```

Q6. Write a program to Identify the given number is an even number or odd number?

Ans.

```
//Identify the given number is an even number or odd number
class TestYourKnowledge6
{
 public static void main(String args [])
 {
     int NO=82;
     if (NO%2==0)
     System.out.println("Number is even");
     else
     System.out.println("Number is odd");
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge6.java
D:\code\chapter2>java TestYourKnowledge6
```

Output:

```
Number is even
```

Q7. Write a program to identify the given number is a perfect square root.

Ans.

```
//Identify the given number is a perfect square root.
import java.lang.Math;
class TestYourKnowledge7
{
 public static void main(String args [])
 {
     int NO=3600,K;
     K=(int) Math.sqrt(NO);
     if (NO==K*K)
     System.out.println("Number is perfect square");
     else
     System.out.println("Number is not perfect square");
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge7.java
D:\code\chapter2>java TestYourKnowledge7
```

Output:

```
Number is perfect square
```

Using Multiple if-else Statement

If you need to check more than one condition in your program, you need to use multiple `if-else` in your program. Following is the syntax of multiple `if-else` statements:

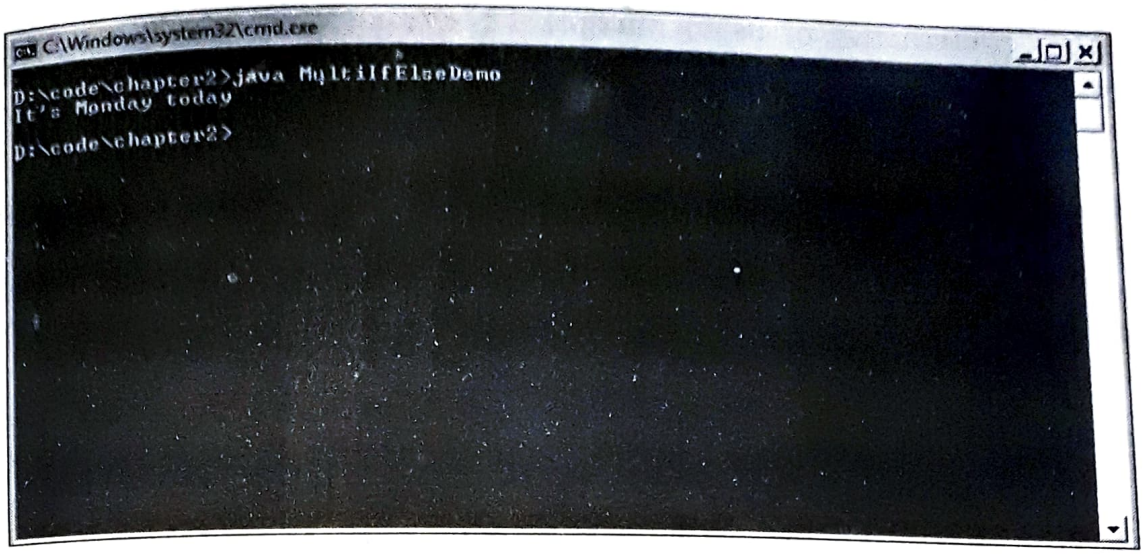
```
if (conditional expression)
  <Statement1 to be executed in case it's true>;
else
  if (conditional expression)
  <Statement2 to be executed in case it's false>;
else
  <Statement2 to be executed in case it's false>;
```

In the preceding syntax, if the condition specified in the first `if` condition is `false`, the control goes to next `if` statement. If the next `if` statement results in `false`, the statements written inside the else block executes. Listing 18 shows how to use multiple `if-else` in your program (you can find the `MultiIfElseDemo.java` file program on the CD in the code\chapter2 folder):

▶ **Listing 18: Using multiple if-else Statement**

```
class MultiIfElseDemo
{
  public static void main(String[] args)
  {
  int i=1;
  if(i==1)
  {
      System.out.println("It's Monday today");
  }
  else
  if(i==2)
  {
      System.out.println("It's Tuesday today");
  }
  else
  if(i==3)
  {
      System.out.println("It's Wednesday today");
  }
  else
  if(i==4)
  {
      System.out.println("It's Trhusday today");
  }
  else
  if(i==5)
  {
      System.out.println("It's Friday today");
  }
  else
  System.out.println("Not a working day!!!");
  }
}
```

In Listing 18, the `MultiIfElseDemo` class is created with multiple `if-else`. The conditions of all the if statements are evaluated until one results to be `true`. If none of if statement returns true result, the else block is executed. Here, the first `if` statement evaluates to be true and shows result as shown in Figure 19:



▲ **Figure 19: Output of the MultiIfElseDemo Program**

In Figure 19, the statement inside the first `if` block is displayed as output.

TEST YOUR KNOWLEDGE

Q8. Write a program to display the biggest number out of three given numbers using if and else statements.

Ans.

```
//Program to display biggest number out of three given numbers using
if
 and
 //else statements
class TestYourKnowledge8
{
 public static void main(String args [])
 {
     int a=19,b=26,c=11;
     if(a>b)
     if(a>c)
     System.out.println("BIG="+a);
     else
     System.out.println("BIG="+c);
     else
     if(b>c)
     System.out.println("BIG="+b);
     else
     System.out.println("BIG="+c);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge8.java
D:\code\chapter2>java TestYourKnowledge8
```

Output:

```
BIG=26
```

Another alternative of using multiple `if-else` statements is `switch` statement in Java. Let's discuss `switch` statement in following section.

Using switch Statement

When there are several options and we have to choose only one option from them, we can use `switch` statement. Depending on the selected option, a particular task is performed. Task here represents one or more statements. Following is the general syntax for using `switch` statement:

```
switch (variable)
{
  case value 1: statements 1;
  case value 2: statements 2;
  case value 3: statements 3;
  :
  case value n : statements n;
  [default : default_statements;]
}
```

In the preceding syntax, depending on the value of the variable, a particular task (statement) is executed. If the variable value is equal to `value1`, `statements1` will be executed. If the variable value is `value2`, `statements2` will be executed, and so on. If the variable value is not equal to `value1`, `value2`, ... then none of the statements will execute. In that case, control comes to the `default` clause and the set of `default_statements` gets executed. In the preceding syntax, the default statement is enclosed in square bracket, [ ] because it is optional. It should be remembered that you cannot use all data types with switch statement. We can use `char`, `int`, `byte`, and `short` types only. Listing 19 shows how to use switch statement in your program (you can find the `SwitchDemo.java` file on the CD in the code\chapter2 folder):

▶ ***Listing 19: Using switch Statement***

```
class SwitchDemo
{
  public static void main(String args[])
  {
      char country='i'; // country is set to 'i'
      switch(country) {
          case 'a': System.out.println("America");
          case 'c': System.out.println("Canada");
          case 'i': System.out.println("India");
          case 'e': System.out.println("England");
          default: System.out.println("No Country");
      }
  }
}
```

In Listing 19, a variable of char type named `country` is declared and is initialized to `i`. Depending on the value stored in the variable `country`, `America` is displayed when the value is `a`; `Canada` is displayed when the value is `c`; `India` is displayed when the value is `i`; and

England is displayed when the value is e. If the value of a country is neither of the specified values—a, c, i, or e, then the value in the default statement is displayed. Figure 20 shows the output of Listing 19:

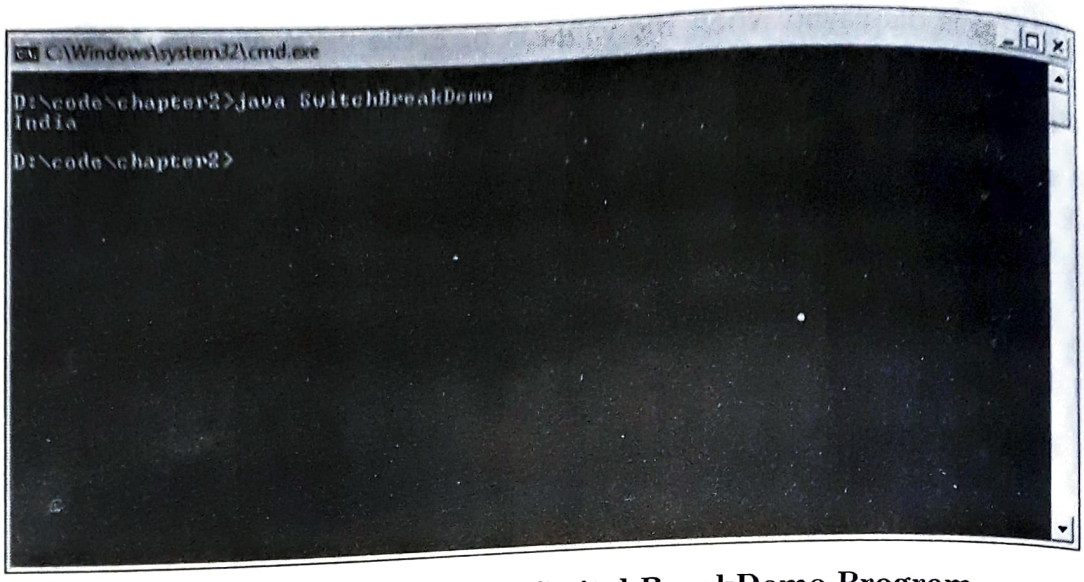
▲ **Figure 20: Output of the SwitchDemo Program**

The output shown in Figure 20 is not as expected. We expected that it would display India, but it is displaying every statement from India onwards. The question here is what could possibly be the reason? The value of variable country is set to 'i', so India should get displayed but we found that it has executed the rest of the statements following India. The reason for this problem is that once India is displayed, the control was not able to come out of the switch statement. For this purpose, break statement should be used in the program (Listing 19), which allows the program to come out of the switch statement after displaying India, as shown in Listing 20 (you can find the SwitchBreakDemo.java file program on the CD in the code\chapter2 folder):

▶ **Listing 20: Using switch break Statement**

```
class SwitchBreakDemo {
 public static void main(String args[])   {
     char country='i';// country is set to 'i'
     switch(country) {
         case 'a': System.out.println("America");
             break;
         case 'c': System.out.println("Canada");
             break;
         case 'i': System.out.println("India");
             break;
         case 'e': System.out.println("England");
             break;
         default: System.out.println("No Country");
     }
 }
}
```

In Listing 20, 'i' becomes true and hence, JVM displays India, as shown in Figure 21 and then executes break, which terminates the switch block:



▲ **Figure 21: Output of the SwitchBreakDemo Program**

In Figure 21, only India is displayed this time. It is because of the use of break statement.

TEST YOUR KNOWLEDGE

Q9. Write a program to demonstrate use of switch statement.

Ans.

```
class TestYourKnowledge9
{
 public static void main(String ar[])
 {
     char grade='A';
     switch(grade)
     {
         case 'A':
             System.out.println("You got more than 90%");
             break;
         case 'B':
             System.out.println("You got more than 80%");
             break;
         case 'C':
             System.out.println("You got more than 60%");
             break;
             default:
             System.out.println("You are fail");
     }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge9.java
D:\code\chapter2>java TestYourKnowledge9
```

Output:

```
You got more than 90%
```

So far, we have learned various selection statements. Let's now take a look on another category of control flow statements used to execute a block of statements repeatedly. These types of control flow statements are known as iteration statements. Let's discuss iteration statements in the following section.

Using Iteration Statements

Repeating the execution of same code fragment several times is called iteration. The basic logic behind iteration is that a sequence of statements is repeated until a certain condition is met. When this condition is false, the iteration terminates, and the loop is completed. There are various types of iteration statements available in Java. The iteration statements are also known as loops in Java. Following are the iteration statements that discussed in this section:

- while loop
- do-while loop
- for loop
- for each loop
- Nested loop
- Let's discuss types of iteration statements in detail now.

Using while Loop

The function of the `while` loop is to execute a specific block of statements until a condition evaluates to be true. Once the condition becomes false, the loop gets terminated. Following is the general syntax for using while loop:

```
while (conditional expression is true) {
  Statements to be executed;
}
```

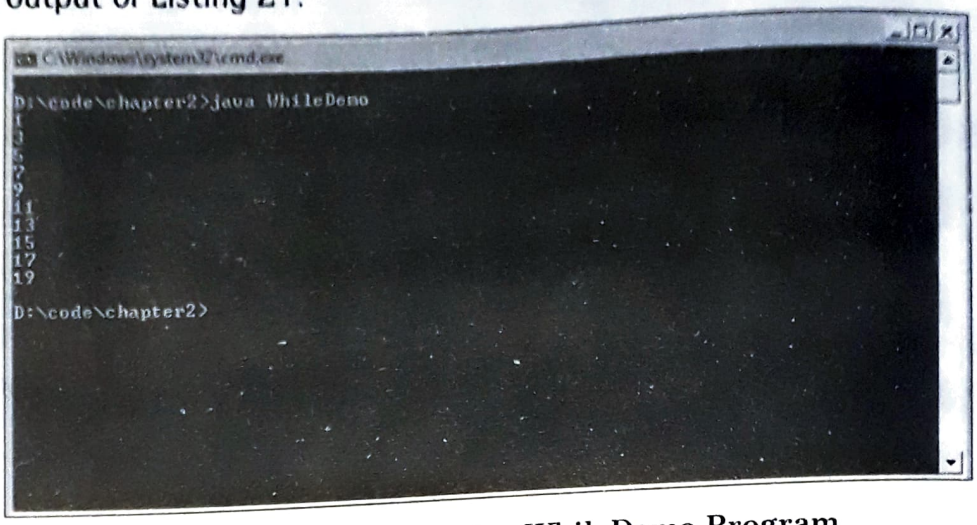
In the preceding syntax, the condition is first tested. If the condition is true, the statements following it will go on executing repeatedly till it continues to be true. Once the condition becomes false, the loop terminates. There must be an increment/decrement statement inside the while block that should make the evaluation result to return false; otherwise, an infinite loop is created. Listing 21 shows how to use while loop in your program (you can find the `WhileDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 21: Using while loop Statement**

```
class WhileDemo {
  public static void main(String args[])   {
      int i=1;// setting the value to 1
      while(i<=20) {
          System.out.println(i);// display i
          i=i+2; // increment the value by 2
      }
  }
}
```

In Listing 21, first ten odd numbers are displayed using while loop. Firstly, variable `'i'` is set to `1`. To display odd numbers, `'i'` gets incremented by `2` in each iteration. When the

evaluation becomes false, that is once the value of 'i' exceeds 20, the loop terminates. Figure 22 shows the output of Listing 21:



▲ Figure 22: Output of the WhileDemo Program

In Figure 22, odd numbers from 1 to 20 is displayed using the while loop. Let's now take a look on another loop called do-while loop, which is almost like a while loop but with a slight difference.

TEST YOUR KNOWLEDGE

Q10. Write a program to print even number up to 10 using while loop.

Ans.

```
class TestYourKnowledge10{
 public static void main(String ar[]) {
     int i=2;
     while(i<=10) {
         System.out.print(i+" ");
         i=i+2;
     }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge10.java
D:\code\chapter2>java TestYourKnowledge10
```

Output:

```
2 4 6 8 10
```

Q11. Write a program to print all even integers divisible by 6 or 8 upto 50.

Ans.

```
import java.lang.Math;
class TestYourKnowledge11
{
 public static void main(String args [])
 {
     int I=2;
```

```
        while (I<=50)
        {
            if (((I%6==0)||(I%8==0)))
            System.out.print(" "+I);
            I+=2;
        }
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge11.java
D:\code\chapter2>java TestYourKnowledge11
```

Output:

```
6 8 12 16 18 24 30 32 36 40 42 48
```

Q12. Write a program to find the given number is prime number or not.

Ans.

```
class TestYourKnowledge12
{
  public static void main(String args[])
  {
      int NO=5;
      int DV=2;
      int K=(int)Math.sqrt(NO);
      while ((NO%DV!=0)&&(DV<=K))
      ++DV;
      if (DV>K)
      System.out.println("The Given no. is Prime No.");
      else
      System.out.println("The given no. is not Prime No.");
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge12.java
D:\code\chapter2>java TestYourKnowledge12
```

Output:

```
the Given no. is Prime No.
```

Q13. Write a program to find the given number is a Fibbonacc series number or not.

Ans.

```
class TestYourKnowledge13
{
  public static void main (String args [])
  {
      int NO=279;
      int f1=1,f2=1;
      while (NO>f1)
```

```
        {
            int  f=f1+f2;
            f1=f2;
            f2=f;
        }
        if
        (NO==f1)
        System.out.println("The term is fibbonacc");
        else

        System.out.println("The term is not fibbonacc");
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge13.java
D:\code\chapter2>java TestYourKnowledge13
```

Output:

```
The term is not fibbonacc
```

Unlike while loop, the statement code will always be executed at least once by the `do-while` loop as the conditional expression is evaluated at the end of the loop.

Using do-while Loop

The `do-while` loop works like a while loop where a group of statements is executed as long as the condition is true. If the condition becomes false, the flow of execution comes out of the `do-while` loop. Following is the general syntax for using while loop:

```
do{
  statements;
} while(condition);
```

In the preceding syntax, the statement code is followed by the while keyword and a test expression. In case the evaluation returns true, the control goes back to the `do` keyword and continues until the evaluation becomes false. Listing 22 shows how you can use the do-while loop in your program (you can find the `DoWhileDemo.java` file on the CD in the code\chapter2 folder):

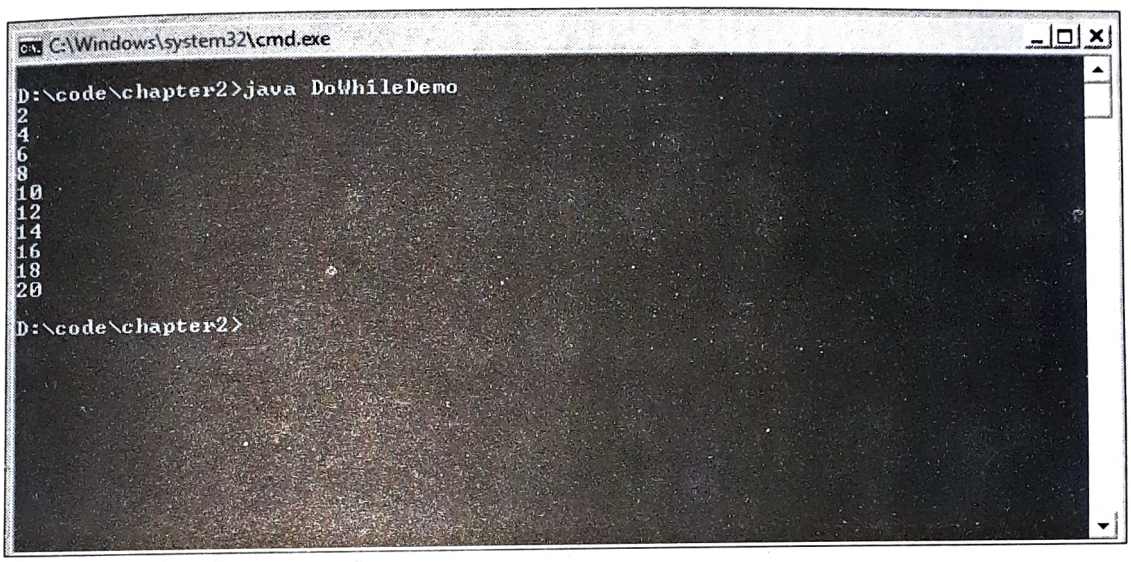
▶ **Listing 22: Using do-while loop Statement**

```
class DoWhileDemo
{
  public static void main(String args[])
  {
      int i=2;// setting the value to 2
      do{
          System.out.println(i);// displaxy i
          i=i+2; // increment the value by 2
      }
      while(i<=20);
```

```
    }
  }
```

In Listing 22, first ten even numbers are displayed using do-while loop in the same way as that of while loop. The difference, however, comes in the manner in which they are implemented. Here, in Listing 22, the value of `i` gets displayed before evaluating the conditional expression unlike `while` loop where the condition is checked first and only then the statements are executed further.

Firstly, variable `i` is set to `2`. To display even numbers, `i` get incremented by `2` in each iteration. The loop gets terminated when the evaluation becomes false, that is once the value of `i` exceeds 20, as shown in Figure 23:



▲ **Figure 23: Output of the DoWhileDemo Program**

In the Figure 23, the `do-while` loop is used to display even numbers up to 20.

TEST YOUR KNOWLEDGE

Q14. Write a program to demonstrate the use of do-while loop.

Ans.

```
class TestYourKnowledge14
{
 public static void main(String args[])
 {
     int i=1;
     do
     {
         System.out.print(i+" ");
         i=i+2;
     }while(i<10);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge14.java
D:\code\chapter2>java TestYourKnowledge14
```

Output:

```
1 3 5 7 9
```

Q15. Write a program to display sum of digits of a given number.

Ans.

```
class TestYourKnowledge15
{
  public static void main(String args[])
  {
      int no=3452;
      int d,sum=0;
      do
      {
          d=no%10;
          sum+=d;
          no=no/10;
      }
      while(no>0);
      System.out.println("Sum of given number is" +sum);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge15.java
D:\code\chapter2>java TestYourKnowledge15
```

Output:

```
Sum of given number is14
```

After learning the basic difference between while and do-while loop, let's now discuss yet another type of iteration statement called for loop.

Using for Loop

The `for` loop is also like a `while` loop, but is syntactically more compact. The `for` loop executes a group of statements as long as a condition is true. Following is the general syntax of the `for` loop:

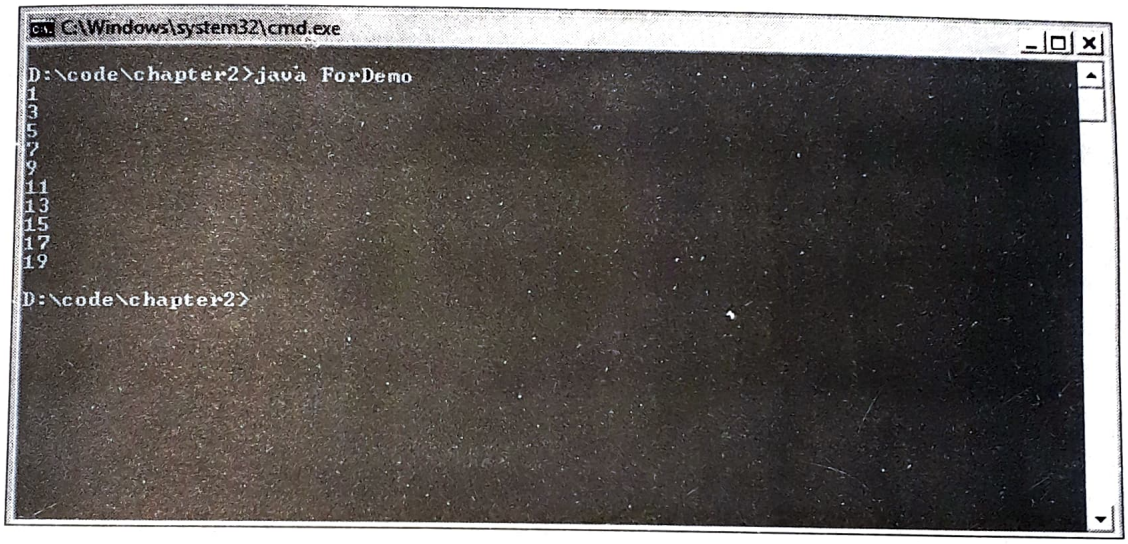
```
for (initializer; condition; increment)
{
  Statements;
}
```

In the preceding syntax, `initializer` is used to set the initial value for the counter to control the iteration. For this purpose, a variable may be declared. A `boolean` condition is evaluated at each pass of the loop to check whether the loop will continue to run or not. In case the condition evaluates to true, the loop will continue; otherwise, it will terminate. Listing 23 shows how you can use for loop in your program (you can find the `ForDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 23: Using for loop Statement**

```
class ForDemo
{
 public static void main(String args[])
 {
     int i=1;
     //setting the value to 1
     for(i=1;i<=20;i=i+2)
     {
         System.out.println(i);
     }
 }
}
```

In Listing 23, the first ten odd numbers are displayed with the help of for loop. Firstly, variable `i` is set to `1`. To display odd numbers, `i` gets incremented by `2` in each iteration. The evaluation becomes false once the value of `i` exceeds 20 and the loop terminates, as shown in Figure 24:



▲ **Figure 24: Output of the ForDemo Program**

In Figure 24, the odd numbers up to 20 is displayed with the help of for loop.

TEST YOUR KNOWLEDGE

Q16. Write a program to print table of 2 with the help of for loop.

Ans.

```
class TestYourKnowledge16
{
 public static void main(String ar[])
 {
     int i=2;
     for(int j=1;j<=10;j++)
     {
         System.out.print(i*j+" ");
     }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge16.java
D:\code\chapter2>java TestYourKnowledge16
```

Output:

```
2 4 6 8 10 12 14 16 18 20
```

Q17. Write a program to display even number up to 20, except 16.

Ans.

```
class TestYourKnowledge17
{
  public static void main(String ar[])
  {
      for(int i=2;i<=20;i+=2)
      {
          if(i==16)
          continue;
          System.out.print(i+" ");
      }
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge17.java
D:\code\chapter2>java TestYourKnowledge17
```

Output:

```
2 4 6 8 10 12 14 18 20
```

Q18. Write a program to display natural number up to 20.

Ans.

```
//Program to display natural numbers up to 20
class TestYourKnowledge18
{
  public static void main(String args[])
  {
      for (int I=1; I<=20; I++)
      System.out.print(" "+I);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge18.java
D:\code\chapter2>java TestYourKnowledge18
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Q19. Write a program to find the sum of natural numbers up to 20.

Ans.

```
//Program to find the sum of natural numbers up to 20.
class TestYourKnowledge19
{
  public static void main(String args[])
  {
      int sum=0;
      for (int I=1;I<=20;I++)
      sum=sum+I;
      System.out.println("Sum of number is " +sum);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge19.java
D:\code\chapter2>java TestYourKnowledge19
```

Output:

```
Sum of number is 210
```

Q20. Write a program to display odd numbers up to 50.

Ans.

```
//Program to display odd numbers up to 50.
class TestYourKnowledge20
{
  public static void main (String args [])
  {
      for(int I=1; I<50; I=I+2)
      System.out.print(" "+I);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge20.java
D:\code\chapter2>java TestYourKnowledge20
```

Output:

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
```

Q21. Write a program to find the sum of a given series X+X2+X3+.....+Xn.

Ans.

```
//Program to find the sum of a given series X+X2+X3+.......+Xn.
class TestYourKnowledge21
{
  public static void main (String args[])
  {
      float X=1.0f; // X value is defined by user
      float sum=0.0f;
```

```
        float term=X;
        for (int I=1;I<=20;I++)
        {
            sum=sum+term;
            term=term*X;
        }
    System.out.println("The Sum of given Expression is "+sum);
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge21.java
D:\code\chapter2>java TestYourKnowledge21
```

Output:

```
The Sum of given Expression is 20.0
```

Q22. Write a program to find sum of the given series X/1+X2/2+.... up to 20 terms.

Ans.

```
//Program to find sum of the given series X/1+X2/2+.... up to 20 terms
class TestYourKnowledge22
{
 public static void main (String args[])
 {
     float K=1.0f;
     float X=1.0f;
     float sum=0.0f;
     float term=X;
     for (int I=1;I<20;I++)
     {
         sum=sum+term;
         term=term*X/K;
         ++K;
     }
     System.out.println("The sum of expression is"+sum);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge22.java
D:\code\chapter2>java TestYourKnowledge22
```

Output:

```
The sum of expression is2.718282
```

Q23. Write a program to display all the natural numbers up to 50 which are not divisible by 5.

Ans.

```
//Program to display all the natural numbers up to 50 which are not
    //divisible by 5.
```

```
class TestYourKnowledge23
{
 public static void main (String args[])
 {
     for (int I=1;I<=49;I++)
     {
         if (I % 5!=0)
         System.out.print(" "+I);
     }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge23.java
D:\code\chapter2>java TestYourKnowledge23
```

Output:

```
1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19 21 22 23 24 26 27 28 29 31 32
33 34 36 37 38 39 41 42 43 44 46 47 48 49
```

Q24. Write a program to find the sum of the series X+A/2!-X+2A/3!+.... UPTO 20.

Ans.

```
//Program to find the sum of the series X+A/2!-X+2A/3!+.... UPTO 20
class TestYourKnowledge24
{
 public static void main(String args[])
 {
     int sign=1;
     float sum=0.0f;
     float A=1.75f,X=2.0f,fact=2.0f;
     float term=X;
     for(int I=1;I<=20;I++)
     {
         term=(X+I*A)/fact*sign;
         sum+=term;
         if(I==10)
         System.out.println("Sum of 10 terms is "+sum);
         sign=-sign;
         fact=fact*(I+2);
     }
     System.out.println("Sum of 20 terms is "+sum);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge24.java
D:\code\chapter2>java TestYourKnowledge24
```

Output:

```
Sum of 10 terms is 1.1981808
Sum of 20 terms is 1.1981808
```

Q25. Write Program to find the sum of following series:

```
21*26        30*35        39*44
-----   +    -----   +    ------ +     UPTO 10 terms
14*16        19*21        24 *26
```

Ans.

```
//Program to find the sum of following series:
/* 21*26    30*35    39*44
   ------ + ----- + ------ + UPTO 10 terms
   14*16    19*21    24 *26
*/

import java.lang.Math;
class TestYourKnowledge25
{
 public static void main(String args[])
 {
     float sum=0.0f;
     float term=0.0f;
     int L=21,K=14;
     for (int I=1;I<=10;I++)
     {
         term=((float)L*(L+5)/(K*K+2));
         sum+=term;
         L+=9;
         K+=5;
     }
     System.out.println("Sum of expression is " +sum);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge25.java
D:\code\chapter2>java TestYourKnowledge25
```

Output:

```
Sum of expression is 30.229387
```

Q26. Write a program to find sum of 16 terms of the form

$x + x^2 + x^3 + x^4 + x^5 \ldots\ldots$

Ans.

```
// Program to find sum of 16 terms of the form
// x +x² + x³ + x⁴ + x⁵...
class TestYourKnowledge26
{
 public static void main (String args [ ])
 {
     float X=(float)1.75;
     int N=16;
     float SUM=(float) 0.0,I;
     float TERM= (float)X;
     for (I=1;I<=N;I++)
     {
         SUM=SUM+TERM;
```

```
            TERM= TERM*X;
        }
        system.out.println("SUM="+SUM);
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge26.java
D:\code\chapter2>java TestYourKnowledge26
```

Output:

```
SUM=18052.172
```

Q27. Write a program to evaluate x/1 + (x sq)/2 + (x cube)/3+ x 4/ 4....up to 16terms

Ans.

```
//Program to evaluate x/1 + (x sq)/2 + (x cube)/3+   x 4/ 4....upto
 16terms
import java.lang.Math;
class TestYourKnowledge27
{
 public static void main(String args [])
 {
     float X=1.75f, N=16 , I;
     float SUM= 0 ;
     int K =2;
     float TERM=X ;
     for (I=1;I<=N;I++)
     {
         SUM=(SUM+TERM);
         TERM = TERM*X/K ;
         ++K ;
     }
     System.out.print("SUM="+SUM);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge27.java
D:\code\chapter2>java TestYourKnowledge27
```

Output:

```
SUM=4.7546024
```

Q28. Write a program to display the sum of 10th+18th+27th TERM of series ((x+a)/2!)-((x+2a)/3!)+((x+3a)/4!)....

Ans.

```
import java.lang.Math;
//SUM OF 10th+18th+27th TERM of series
//((x+a)/2!)-((x+2a)/3!)+((x+3a)/4!)....
class TestYourKnowledge28
{
 public static void main(String args[])
```

```
        {
            int SIGN=1;
            double SUM=0.0,TERM,A=1.75,FACT=2.0,X=1.9;
            for (int I=1;I<=27;I++)
            {
                TERM=(X+I*A)/FACT*SIGN;
                SUM+=TERM;
                SIGN=-SIGN;
                FACT=FACT*(I+2);
                if(I==10)
                System.out.println("Sum of 10 terms ="+SUM);
                if(I==18)
                System.out.println("Sum of 18 terms ="+SUM);
            }
            System.out.println("Sum of 27 terms ="+SUM);
        }
    }
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge28.java
D:\code\chapter2>java TestYourKnowledge28
```

Output:

```
Sum of 10 terms =1.1613928533850408
Sum of 18 terms =1.1613928941256921
Sum of 27 terms =1.1613928941256921
```

Q29. Write a program to calculate Ncr.

Ans.

```
class TestYourKnowledge29
{
 static int fact(int n)
 {
     int f=1;
     for(int I=1;I<=n;I++)
     f=f*1;
     return(f);
 }
 public static void main(String args[])
 {
     int ncr;
     ncr=fact(3)/fact(2)/fact(3-2);
     System.out.println("ncr="+ncr);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge29.java
D:\code\chapter2>java TestYourKnowledge29
```

Output:

```
ncr=1
```

Q30. Write a program for X+X^2+X^3+..... series.

Ans.

```
class TestYourKnowledge30
{
 public static void main(String args[])
 {
     int N=16;
     float sum=0.0f;
     float X=2.00f;
     float term=X;
     for(int I=1;I<=N;I++)

     {
         sum=term;
         term=term*X;
     }
     System.out.println("sum="+sum);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge30.java
D:\code\chapter2>java TestYourKnowledge30
```

Output:

```
sum=65536.0
```

Q31. Write a program for X/1!+(1+2X)/2!+(1+3X)/3!+..... series.

Ans.

```
class TestYourKnowledge31
{
 public static void main(String args[])
 {
     int N=21,F=2;
     float Sum=0.0f,X=1.75f;
     float term =X;
     for(int I=1;I<=N;I++)
     {
         Sum=Sum+term;
         term=(1+(I+1)*X)/F;
         F=(F*(I+2));
     }
     System.out.println("Sum=  " +Sum);
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge31.java
D:\code\chapter2>java TestYourKnowledge31
```

Output:

```
Sum=  5.4752755
```

Q32. Write a program for calculating sum of ((1+2A)/2!)+(1+3A)/3!)+(1+4A)/4!)... series.

Ans.

```
import java.lang.Math;
class TestYourKnowledge32
{
  public static void main(String args[])
  {
      float SUM=0.0f,TERM=1.0f,A=1.75f,FACT=2.0f;
      int I;
      for (I=1;I<=20;I++)
      {
          TERM=((1+(I+1)*A)/FACT);
          SUM=+TERM;
          FACT=FACT*(I+2);
      }
      System.out.println("SUM ="+SUM);
  }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge32.java
D:\code\chapter2>java TestYourKnowledge32
```

Output:

```
SUM =7.3887857E-19
```

Q33. Write a program for calculating T=SUMMTN Xi square/K1! +SUMMTN Yi square/K2!.

Ans.

```
class TestYourKnowledge33
{
  static int fact(int N)
  {
      int F=1;
      for(int I=1;I<=N;I++)
      F=F*I;
return(F);
  }
  static float sumXY(int N,int K,int X[])
  {
      float sum=0.0f;
      for (int I=0;I<=N;I++)
      sum=+X[I]*X[I];
      return(sum/fact(K));
  }
  public static void main(String args[])
  {
      int N=5,K1=6,K2=7;
      float T;
      int X[]={1,3,5,8,11,16};
      int Y[]={2,4,8,16,18,20};
      T=sumXY(N,K1,X)+sumXY(N,K2,Y);
      System.out.println("sum="+T);
```

```
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge33.java
D:\code\chapter2>java TestYourKnowledge33
```

Output:

```
sum=0.43492064
```

Q34. Write a program to develop all 3 digit and 4 digit Armstrong numbers, that is, number equal to the sum of cubes of the digits of the number. Example 153 = 1x1x1+5x5x5+3x3x3.

Ans.

```
class TestYourKnowledge34
{
 public static void main (String args[])
 {
     int D,SUM;
     int NO;
     for (int I=100;I<=9999;I++)
     {
         NO= I;
         SUM =0;
         while (NO>0)
         {
             D = NO%10;
             SUM = SUM + D*D*D;
             NO=NO/10;
         }
         if (SUM == I)
         System.out.println(I);

     }
}
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge34.java
D:\code\chapter2>java TestYourKnowledge34
```

Output:

```
153
370
371
407
```

Q35. Write a program to find a five digit Number which on Multiplication by 4 reverses its order.

Ans.

```
class TestYourKnowledge35
{
```

```
    public static void main (String args[])
    {
        for (int NO=10000;;NO++)
        {
            int p = NO*4;
            int sum = 0,d;
            while (p>0)
            {
                d= p%10;
                sum = sum *10+ d;
                p= p/10;
            }

            if (sum == NO)
            {
                System.out.println("Five digit Number which on
                    Multiplication
                    by 4 reverses its order   IS:   "+ NO);
                break;
            }
            else
            ++NO;
    }
    }
    }
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge35.java
D:\code\chapter2>java TestYourKnowledge35
```

Output:

```
Five digit Number which on Multiplication by 4 reverses its order
IS:  21978
```

Q36. Write a program to find all four digit perfect squares, where the number forming by first two digits and number forming by last two digits are also perfect squares.

Ans.

```
class TestYourKnowledge36
{
 public static void main (String args[])
 {

    for (int i=1000;i<9999;i++)
    {
        int k =(int) Math.sqrt(i);
        if (k*k == i)
        {
            int LN =i/100;
            int RN = i%100;
            int LK = (int) Math.sqrt(LN);
            int RK = (int) Math.sqrt(RN);
            if((LK *LK == LN) && (RK *RK == RN))

            System.out.println(i);
```

```
            }
        }
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge36.java
D:\code\chapter2>java TestYourKnowledge36
```

Output:

```
1600
1681
2500
3600
4900
6400
8100
```

The next type of iteration statement that we are going to discuss is for-each loop. This loop executes statements for each element of the collection.

Using for-each Loop

The for-each loop is specifically designed to handle the elements of a collection. A collection is represented by a group of objects. Array is an example of collection since it stores a group of objects, such as integer values or strings. Similarly, the classes of `java.util package` are also developed to handle a group of objects. The for-each loop repeatedly executes a group of statements for each element of the collection. Following is the general syntax for using `for-each` loop:

```
for(var:collection)
{
  Statements:
}
```

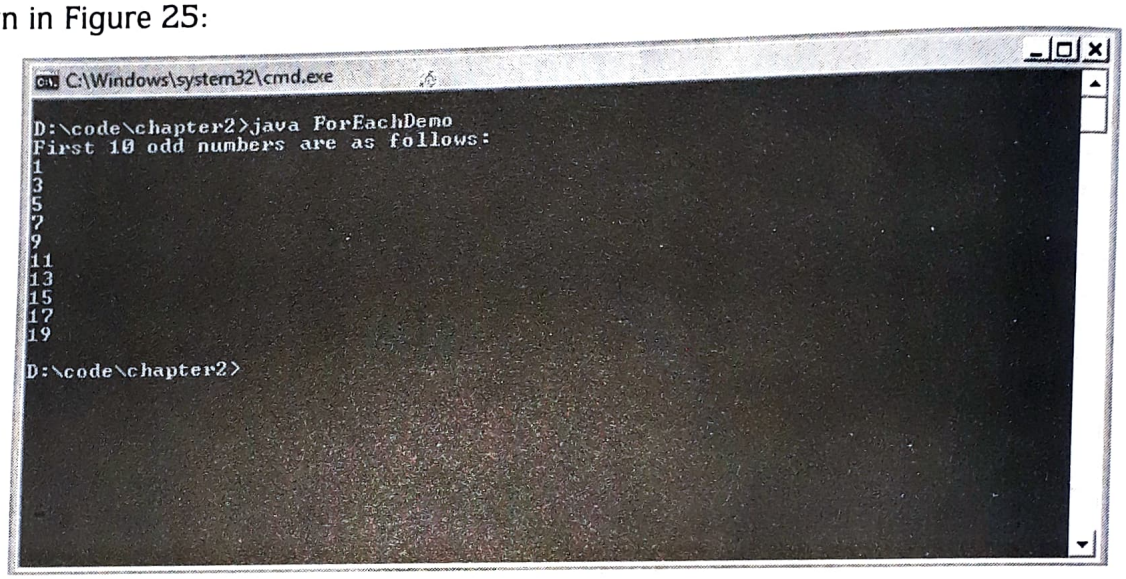
In the preceding syntax, `var` represents each element of the collection one by one. For instance, there is a collection of `5` elements then this loop will execute `5` times and the `var` will represent these elements one by one. Let's understand the use of `for-each` loop with the help of an example shown in Listing 24 (you can find the `ForEachDemo.java` file on the CD in the code\chapter2):

▶ **Listing 24: Using for-each Loop**

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

```
        int arr[]={1,3,5,7,9,11,13,15,17,19}; //declare an array with 10 elements
        System.out.println("First 10 odd numbers are as follows:");
        for(int i:arr) // use for each to retrieve elements from an array
        {
            System.out.println(i); // i represents each element of array
        }
        }
    }
```

In Listing 24, the `for-each` loop is used to display each element of an array. Here, there is a collection of array type containing ten elements. The `for-each` loop will execute `10` times and `i` will represent these elements thereby executing statements for each of these elements, as shown in Figure 25:



▲ **Figure 25: Output of the ForEachDemo Program**

In Figure 25, the numbers in the array are displayed using the `for-each` loop. Apart from using single loops in your program, you can also use nested loops in you program. Let's discuss nested loops in the following section.

➡ Working with Nested Loop

It is possible to nest conditional statements as well as iteration statements in Java. Nested statements can be conditional statements within another conditional statements (nested if statements) or it can also be iteration statements containing other iteration statements (nested for loop, while loop, of do-while loop).

A nested for loop is the `for loop` within another for loop. Here, execution starts from the outer `for` loop and the control then passes to the inner `for` loop. The inner `for` loop continues to execute till the condition evaluates to true. Once the inner loop is completed, the flow of execution is passed to outer loop for the next iteration and executes till the evaluation returns true. Let's understand the use of nested `for` loop with the help of an example shown in Listing 25 (you can find the `NestedLoopDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 25: Using nested loop Statement**

```
class NestedLoopDemo
{
```

```
    public static void main(String args[])
    {
        int num1,num2;
        for(num2 = 0; num2 <= 3;  num2++)
        {
            for(num1 = 0; num1 <= 2; num1++)
            {
                System.out.println(num2 + "    " + num1);
            }
        }
    }
}
```

In Listing 25, executions begin with the outer `for` loop in which `num2` is initialized to `0`. Now, the execution enters into the inner `for` loop where `num1` is set to `0`. The inner `for` loop continues to execute until the condition becomes false. In this way, the inner loop is executed three times, with `num1` changing its value from 0 to 2.

Once the inner `for loop` is completed, then the execution goes to the outer `for loop` and `num2` is set to `1`. This time, the execution again comes into the inner `for` loop and the statements following this loop will execute again for three times. Again, the control goes back to the outer `for` loop and `num2` is set to `2`. The outer loop executes till its value exceeds `3`. In this way, the outer loop executes for `4` times.

In Listing 9, the outer `for loop` is executed `4` times and the inner `for loop` is executed `3` times for each value of `num2` and hence inner loop is executed `12` times in total, as shown in Figure 26:

▲ **Figure 26: Output of the NestedLoopDemo Program**

Implementation of nested for loops is shown in Figure 26. Here, a `for` loop is used within another `for` loop. The outer `for` loop is first executed. Variable for controlling the iteration for this loop is `num2`, which is set to an initial value of `0`. The control soon goes to the inner `for` loop where `num1` is set to `0`. This loop executes till the value of `num1` exceeds `2`. In that case, the inner loop executes `3` times with value of `num2` set to `0` and that of `num1` as `0`, `1`, and `2` respectively in all the three executions.

The flow of execution is then passed back to the outer loop where the value of num2 is now set to 1 after undergoing increment. Again, the control is passed to the inner loop and it will execute for 3 times with the value of num2 set to 1 and that of num1 as 0, 1, and 2 respectively. The outer loop continues to execute until the value of num2 exceeds 3 and the condition becomes false. In that case, the outer loop continues to execute as long as the value of num2 is equal to 3. Similar to the for loop nesting, other loops can also be nested.

TEST YOUR KNOWLEDGE

Q37. Write a program to print following number pyramid.

```
/*
 1

 1 2

 1 2 3

 1 2 3 4

 1 2 3 4 5

 1 2 3 4 5 6

 1 2 3 4 5 6 7

 1 2 3 4 5 6 7 8

 1 2 3 4 5 6 7 8 9

 1 2 3 4 5 6 7 8 9 10
 */
```

Ans.

```
// Program to print following number pyramid
/*
 1

 1 2
 1 2 3

 1 2 3 4

 1 2 3 4 5

 1 2 3 4 5 6

 1 2 3 4 5 6 7

 1 2 3 4 5 6 7 8

 1 2 3 4 5 6 7 8 9

 1 2 3 4 5 6 7 8 9 10
 */

class TestYourKnowledge37
{
```

```
    public static void main(String args[])
    {
        for (int K=1;K<=10;K++)
        {
            system.out.println(" ");
            for(int I=1;I<=K;I++)
            system.out.print(" "+I);

        }
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge37.java
D:\code\chapter2>java TestYourKnowledge37
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

Q38. Write a program to display pyramid of numbers:

```
10 9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
7 6 5 4 3 2 1
6 5 4 3 2 1
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

Ans.

```
//Program to display pyramid of numbers:
/*10 9 8 7 6 5 4 3 2 1
  9 8 7 6 5 4 3 2 1
  8 7 6 5 4 3 2 1
  7 6 5 4 3 2 1
  6 5 4 3 2 1
  5 4 3 2 1
  4 3 2 1
  3 2 1
  2 1
  1*/
class TestYourKnowledge38
{
  public static void main(String args [])
```

```
        {
            for (int K=10;K>=1;K--)
            {
                System.out.println(" ");
                for (int I=K;I>=1;I--)
                System.out.print(" "+I);
            }
        }
    }
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge38.java
D:\code\chapter2>java TestYourKnowledge38
```

Output:

```
10 9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
7 6 5 4 3 2 1
6 5 4 3 2 1
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

Q39. Write a program to create a Floyd triangle of numbers.

Ans.

```
class TestYourKnowledge39
{
 public static void main(String args[])
 {
        int i,j;
    for(i=1;i<=10;i++)
    {
        System.out.println(" "+i);
        for(j=i+1;j<=10;j++)
        System.out.print(" " +j);
    }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge39.java
D:\code\chapter2>java TestYourKnowledge39
```

Output:

```
1
2 3 4 5 6 7 8 9 10 2
3 4 5 6 7 8 9 10 3
4 5 6 7 8 9 10 4
```

```
5 6 7 8 9 10 5
6 7 8 9 10 6
7 8 9 10 7
8 9 10 8
9 10 9
10 10
```

Q40. Write a program to print I,J,K,L such that I<J<K<L,L=I+J+K+,I,J,K,<100.

Ans.

```
class TestYourKnowledge40
{
 public static void main(String args[])
 {
     int I=0,J=0,K=0,L=0;
     for (I=1;I<=5;I++)
     for (J=I+1;J<=5;J++)
     for  (K=J+1;K<=5;K++)
     {
         L=I+J+K;
         if (L<=15)
         System.out.println(" "+I+" "+J+" "+K+" "+L);
 }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge40.java
D:\code\chapter2>java TestYourKnowledge40
```

Output:

```
1 2 3 6
1 2 4 7
1 2 5 8
1 3 4 8
1 3 5 9
1 4 5 10
2 3 4 9
2 3 5 10
2 4 5 11
3 4 5 12
```

Q41. Write a program to generate Number Pyramid.

Ans.

```
class TestYourKnowledge41
{
 public static void main (String args[])
 {
     int N=10;// Number Rows of Pyramid
     int j=0;
     for (int K = 1; K<=N;K++)
     {
         System.out.println();
         for (int i = 1 ; i<=K;i++)
```

```
            {
                ++j;
                System.out.print (" " + j);
            }
        }
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge41.java
D:\code\chapter2>java TestYourKnowledge41
```

Output:

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55
```

Q42. Write a program for generation and display of spiral matrix of order 5 as shown below:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Spiral matrix of maximum order 10X10

Ans.

```
class TestYourKnowledge42
{
 public static void main(String args[])
 {
     int A[ ][ ]=new int[11][11];
     int M=10; // Number of rows
     int N=9; // number of colums
     int I,J,K,count;
     for(I=1;I<=10;I++)
     for(J=1;J<=10;J++)
```

```
        A[I][J]=0;
        I=1;
        J=1;
        K=0;
        count=0;
        do

        {
            ++count;
            do
            {
                ++K;
                A[I][J]=K;
                if (count %2!=0) ++J;
                else --J;
            }
            while ((J<=N) && (J>=1)&&(A[I][J]==0));
            if (count %2!=0) J--;
            else J++;
            if (count %2!=0) I++;
            else I--;
                do
            {
                ++K;
                if( A[I][J]==0) A[I][J]=K;
                if (count %2!=0) I++;
                else I--;
            }
            while((I<=M)&&(I>=1)&&(A[I][J]==0));
            if (count %2!=0) --I;
            else ++I;
                if (count %2!=0) --J;
                else ++J;
        }

        while(K<M*N);
        for(I=1;I<=M;I++)
        {
            System.out.println("\n");
            for(J=1;J<=N;J++)
            System.out.print("   "+A[I][J]);
        }
        System.out.println("\n");
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge42.java
D:\code\chapter2>java TestYourKnowledge42
```

Output:

```
1 2   3   4   5   6   7   8   9
34    35  36  37  38  39  40  41  10
33    60  61  62  63  64  65  42  11
32    59  78  79  80  81  66  43  12
31    58  77  88  89  82  67  44  13
```

30	57	76	87	90	83	68	45	14
29	56	75	86	85	84	69	46	15
28	55	74	73	72	71	70	47	16
27	54	53	52	51	50	49	48	17
26	25	24	23	22	21	20	19	18

So far, we have discussed about control flow statements, such as selection statements and iteration statements. In the next section, we study another kind of control flow statements known as jump statements.

Using Jump Statements

Jump statements are the statements that transfer control from one part of the program to another. The jump statements supported by Java are as follows:

- break
- continue
- return

Let's discuss the types of jump statements in detail now.

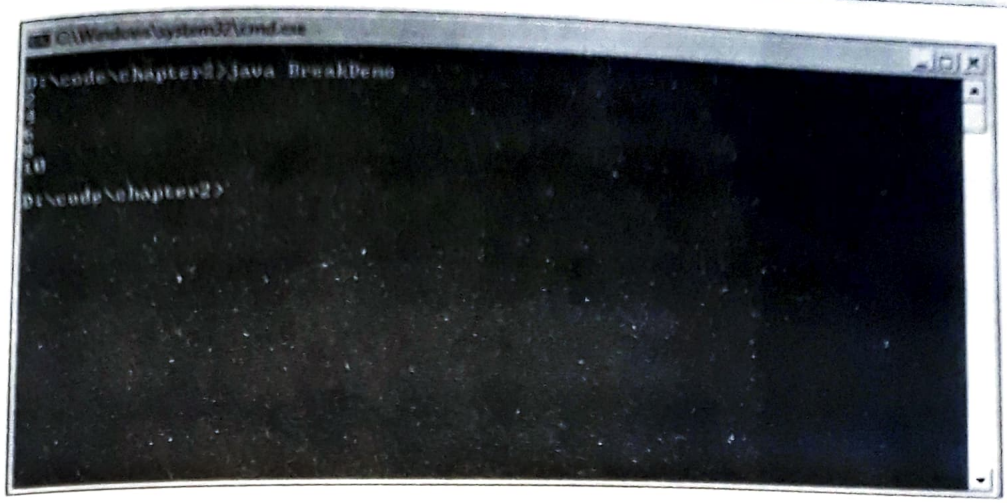
Using break Statement

One of the important purposes of `break` statement is to exit from `switch` statement. At times, we need to break out of a loop before the whole thing gets executed. The `break` statement allows us to get out of a loop even before it is completely executed. This statement is written in the form of a simple `break` keyword followed by a semicolon. Let's understand the use of `break` statement with the help of an example shown in Listing 26 (you can find the `BreakDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 26: Using break Statement**

```
class BreakDemo
{
 public static void main(String[] args)
 {
     int i = 0;
     while (true) // loop will execute infinitely
     {
         i=i+2;
         if (i >10)
         {
         break;    // when the condition becomes true break will get execute
         }
         System.out.println(i);
     }
 }
}
```

In Listing 26, an infinite loop is shown to display even numbers up to infinity. In order to avoid infinity situation, a `break` statement is used within a condition stating that if the value of the `i` variable exceeds `10`, then the control comes out of the `while` loop. If the control comes out of the loop, then the first 5 even numbers are displayed, as shown in Figure 27:



▲ **Figure 27: Output of the BreakDemo Program**

In Figure 27, because of execution of the `break` statement even number up to 10 is displayed and loop comes out of infinite loop.

TEST YOUR KNOWLEDGE

Q43. Write a program to demonstrate the use of break statement in while loop.

Ans.

```
/ class TestYourKnowledge43
{
 public static void main(String[] args)
 {
     int i = 10;
     while (true)
     {
         i=i+5;
         if (i >50)
         {
             break;
         }
         System.out.print(i+" ");
     }
 }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge43.java
D:\code\chapter2>java TestYourKnowledge43
```

Output:

```
15 20 25 30 35 40 45 50
```

The `break` statement can also be used as an alternative to the `goto` statement. Java does not support `goto` statements directly but indirectly the same can be implemented with the help of a `labeled break` statement.

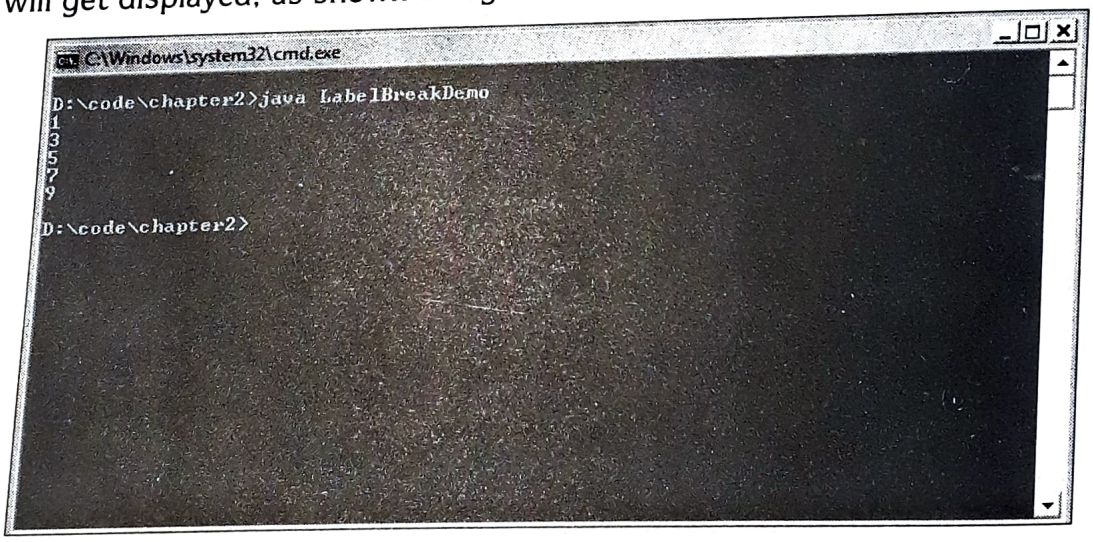
The `break` statement, without a label, transfers control out of the nearest enclosing `for`, `while`, `do`, or `switch` statement. If the `break` keyword is followed by a label of an enclosing

statement, execution transfers out of that enclosing statement. Let's understand the use of a labeled break statement with the help of an example shown in Listing 27 (you can find the `LabelBreakDemo.java` file program on the CD in the code\chapter2 folder):

▶ **Listing 27: Using labeled break Statement**

```
class LabelBreakDemo
{
 public static void main(String[] args)
 {
     int i = 1;
     breaking:while (true) // loop will execute infinitely
     {
         if (i >10)
         {
             break breaking;
         }
         System.out.println(i);
         i=i+2;
     }
 }
}
```

In Listing 27, an infinite loop is shown to display odd numbers up to infinity. In order to avoid this situation, a `labeled break` statement is used within a condition stating that if the value of `i` variable exceeds `10`, the control comes out of the `while` loop and only first 5 odd numbers will get displayed, as shown in Figure 28:



▲ **Figure 28: Output of the LabelBreakDemo Program**

In Figure 28, because of execution of the `break` statement, the `while` loop terminates and the odd number up to 9 is displayed only.

TEST YOUR KNOWLEDGE

Q44. Write a program to demonstrate the use of labeled break.

Ans.

```
class TestYourKnowledge44
{
 public static void main(String[] args)
 {
```

```
        int i = 10;
        breaking:
        while (true)
        {
            if (i >50)
            {
                break breaking;
            }
            System.out.print(i+" ");
            i=i+10;
        }
    }
}
```

Execution:

```
D:\code\chapter2>javac TestYourKnowledge44.java
D:\code\chapter2>java TestYourKnowledge44
```

Output:

```
10 20 30 40 50
```

Using continue Statement

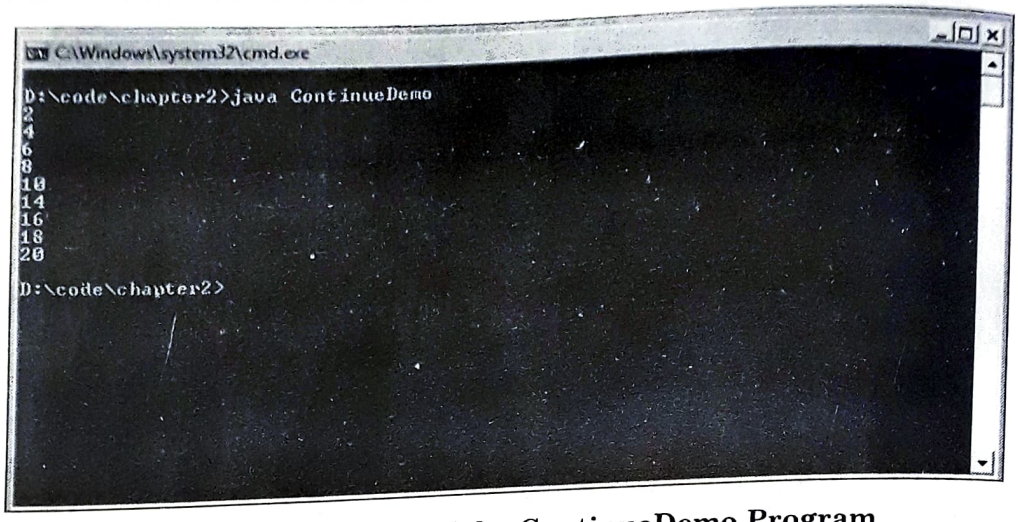
The `continue` statement is used to quit the current iteration of the loop and begin the execution of the next iteration of the loop. When `continue` is used without a label, it starts a new iteration of the innermost loop. When it is used with a label, which represents the name of the containing loop, it starts a new iteration of the named loop. Let's understand the use of `continue` statement with the help of an example shown in Listing 28 (you can find the `ContinueDemo.java` file on the CD in the code\chapter2 folder):

▶ **Listing 28: Using continue Statement**

```
class ContinueDemo
{
 public static void main(String[] args)
 {
 for (int i=2;i<=20;i=i+2)
 {
 if (i == 12) // if i becomes 12 then continue statement will get execute
 {
     continue;
 }
 System.out.println(i);
 }
 }
}
```

In Listing 28, first ten even numbers are shown excluding `12` from the result. This is done with the help of `continue` statement inside the `if` condition, which gets executed in case, the value of `i` variable becomes 12. Here, even numbers are shown successfully until the value of `i` becomes 12. When the value of `i` becomes 12, the control of execution quits from the

current loop and returns back to the `for` loop. The next iteration then begins to display the remaining even numbers till the value of `i` exceeds 20, as shown in Figure 29:



▲ **Figure 29: Output of the ContinueDemo Program**

In Figure 29, because of execution of the continue statement 12 is not displayed and the next iteration of the loop starts.

That was all about the `continue` statement. Let's now discuss about another kind of jump statement available in Java called `return` statement.

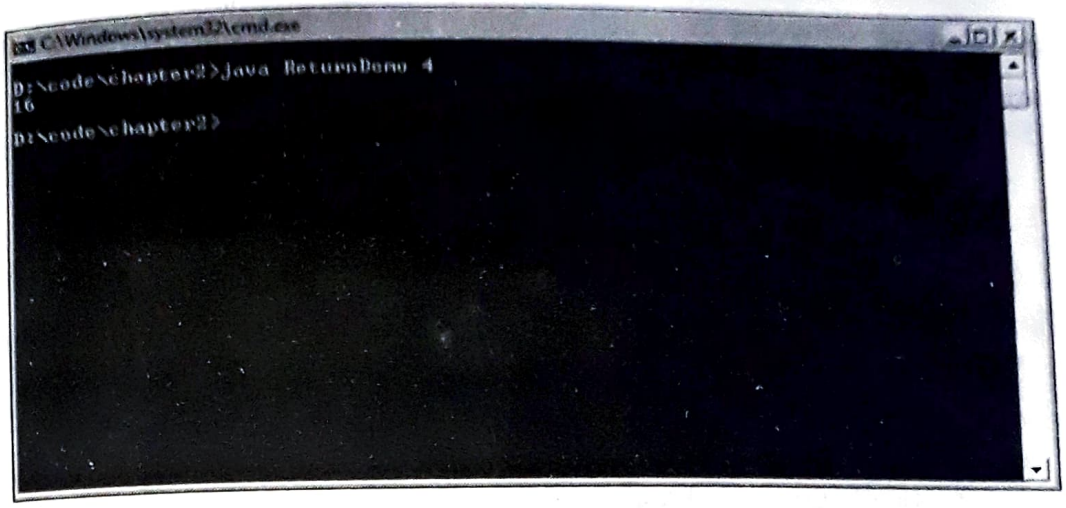
Using return Statement

The `return` statement causes the control of execution to return back to the caller of the method. Let's understand the use of `continue` statement with the help of an example shown in Listing 29 (you can find the `ReturnDemo.java` file on the CD in the code\chapter2 folder):

▶ ***Listing 29: Using return Statement***

```
class ReturnDemo
{
 public static void main(String args[])
 {
     int input=Integer.parseInt(args[0]);
     int result=square(input);
     System.out.println(result);
 }
 public static int square(int x)
 {
     return x*x;
 }
}
```

In Listing 29, the `square()method` accepts an integer value, calculates its square value, and returns that value to the `main()` method, which is its calling method. In the `main()` method, we are calling the `square()` method and passing the input that is supplied by the user at run time. Here, we are directly calling the `square ()` method since it is a static method as static methods can be called directly inside a static context. Figure 30 shows the output of Listing 29:



▲ **Figure 30: Output of the ReturnDemo Program**

In Figure 30, the vale 4 is entered at run time, which is passed to the `square()` method. Therefore, the `square()` method returns 16 as output.

LIBRARY Govt. Women Polytechnic

Summary

This chapter begins with a description of data types available in Java. Then the chapter discusses the members of Java programming such as class, object and methods. You have learned how these members are declared and used in a Java program. The chapter discusses about the type casting also. The Flow control statements section explains the if, if...else, and switch...case statements with the help of numerous examples to show the various ways to set conditions in a program and make decisions based on those conditions. This chapter also discussed how to use the switch...case statement as an alternative for multiple if...else statements to reduce the complexity associated with handling multiple curly braces in a program.

This chapter shows the use and function of the various looping constructs in Java. It begins by explaining the use of the while loop, a very basic loop, which executes a code block until a specified range is reached. This chapter also discusses the for loop, including how this loop is frequently used to replace the while loop. The for-each loop, introduced in Java 5, and used to extract the elements of a collection is also explained in detail.

After discussing various loops, the break, continue, and return statements are discussed to enhance your knowledge on how to control the flow of a program within a loop. The labeled break statement, which replaces the traditional `goto` statement, is shown with a code to give you a clear idea about using this statement in a Java program.

Review Questions

Now, let's solve the following types of questions related to this chapter:

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

■ True or False

State True or False for the following:

1. The long data type is a type of integer data type. **True**
2. The width of the byte data type is 4 bytes. **False**
3. The default value of the boolean data type is false. **True**
4. A class can be declared with the public access specifier. **True**
5. The public access specifier allows any class to access the class members out of that class. **True**
6. A private class member can be accessed in its subclasses. **False**
7. When you don't create any constructor in your class, a default constructor is automatically created. **True**
8. The constructor in a class cannot be overloaded. **False**
9. The methods in a class can be overloaded. **True**
10. A method can be overloaded by changing their return type. **False**
11. Methods are overridden when used inheritance. **True**
12. Objects are always created with the new keyword. **True**
13. Only one if statement can be used in a program. **False**
14. The switch statement can accept any type of data. **False**
15. The do-while loop executes at least once, whether the condition provided in while is true or false. **True**

■ Multiple Choice Questions

Answer the following multiple choice questions:

Q1. Which of the following is/are not a data type in Java?

A. int
B. float
C. double
D. String

Ans. Option D is correct.

Q2. Which of the following is/are correct initialization?

A. int i='A'
B. float f=10f
C. double d=3.25
D. char c="vikash"

Ans. Options B and C are correct.

Q3. Which of the following is/are the incorrect declaration of the sum() method?

A. sum(int a, int b)
B. int sum()
C. void sum(int a, int b)
D. public void sum()

Ans. Option A is correct.

Q4. Which of the following is/are the correct declaration of the while loop?

A.

```
while(int j)
{
 j++;
 system.out.println(j);
}
```

B.

```
int j=1;
while(j<5)
{
 system.out.println(j);
 j++;
}
```

C.

```
while( j)
{
j++;
System.out.println(j);
}
```

D.

```
int j=10;
while(j>=1)
{
System.out.println(j);
j--;
}
```

Ans. Options B and D are correct.

Q5. Which of the following is the correct syntax of do-while loop?

A.

```
do
{
<statements>
}while(condition)
```

B.

```
do
{
 <statements>
}while(condition);
```

C.

```
do
{
 <statements>
```

```
while(condition);
}
```

D.

```
do
while(condition)
{
 <statements>
}
```

Ans. Option B is correct.

Q6. Which of the following is/are valid if statement?

A. if(either x or y)
B. if(a=5)
C. if(a==7)
D. if(a>10)

Ans. Options C and D are correct.

Q7. Which of the following is/are the overloaded method?

A. int sum(int x, int y)
B. float sum(int x, int y)
C. int sum(float x int y)
D. int sum(int y, float x)

Ans. Options A, C, and D are correct.

Q8. Consider the following code snippet:

```
char grade='A';
switch(grade)
{
case 'A':
  System.out.println("Grade A");
case 'B':
  System.out.println("Grade B");
default :
  System.out.println("Fail");
}
```

Select the output of the preceding code snippet from the following outputs.

A. Grade A
B. Grade A
   Grade B
C. Grade A
   Grade B
   Fail
D. Fail

Ans. Option C is correct.

Q9. Which of the following access specifier can protect a field to be accessed in the subclass?

A. public
B. protected
C. private
D. default

Ans. Option C is correct.

Q10. **Consider the following code snippet:**

```
for(int i=1; i<10;i=i+2) {
 system.out.print(i+" ");
}
```

Which of the following is the correct output?

A. 1 2 3 4 5 6 7 8 9 10
B. 1 3 5 7 9 10
C. 1 3 5 7 9
D. 2 4 6 8 10

Ans. Option C is correct.

Q11. **Which of the following statement is used to exit a loop based on some condition?**

A. continue
B. break
C. default
D. return

Ans. Option B is correct.

Q12. **Which of the following control flow statement is not available in Java?**

A. switch
B. if
C. goto
D. if-else

Ans. Option C is correct.

Q13. **Consider the following code snippet:**

```
int i=5;
do
{
 system.out.println(i);
}while(i>5);
```

Which of the following is the correct output of the preceding code snippet?

A. 5 6
B. 5
C. Nothing will be displayed
D. 5 6 7... up to infinity

Ans. Option B is correct.

Q14. **Complete the following statement:**

```
..................... method can be accessed in all the classes and packages.
```

A. public
B. protected
C. private
D. default

Ans. Option A is correct.

Q15. **Which of the following statements is used in methods to return a value?**

A. continue
B. break
C. default
D. return

Ans. Option D is correct.

Short Answer Questions

Q1. What are the main differences between while loop and do-while loop?

Ans. Both the while loop and do-while loop executes a block of codes repeatedly until the specified condition becomes false. The basic difference between while and do-while loop is that the specified condition is checked in the beginning in case of while loop, whereas in case of do-while loop, the specified condition is checked after execution of the enclosed statements. In case of while loop, it is possible that the code inside the loop will not execute even once, if the condition returns false. Whereas, in case of do-while loop, the loop will execute at least once, because the condition is verified at the end of the loop.

Q2. How many data types are available in Java? Describe briefly.

Ans. Java is a strongly typed language, and all the manipulation of data is done along with the associated data type. In Java programming language, you cannot even declare a variable without specifying its data type. The data types available in Java can be divided into four broad categories:

- **Integer:** Represents integers using byte, short, int, and long
- **Floating point:** Represents fractional numbers using float and double data types
- **Character:** Represents char, which is used to store characters
- **Boolean:** Represents logical values using boolean data type

Q3. Explain character data type with example.

Ans. The character data type is used to represent and store the character constants in Java programs. The character data type available in Java is `char`. This datatype follows the Unicode character set to represent the characters that can be uniquely identified by the processor. Unicode is a standard character set that supports all the international characters of various languages, such as American, Latin, and so on. The char has width of 2 bytes and range of 0 to 65535.

An examples of `char` data types is given here:

```
char grade='A';
```

Here, A is a character stored in grade.

Q4. Mention the rules that must be taken care while using identifiers for declaring variables?

Ans. Following are the constraints that must be taken care while declaring variables:

- An identifier must start with an underscore, a letter, or a dollar sign ($)
- An identifier may consist of a letter, a number, an underscore, and dollar sign
- Spaces are not allowed in between an identifier
- Identifiers are case sensitive that means the words salary, Salary, and SALARY are taken as three different identifiers
- Java keywords should not be used while naming an identifier since they hold some special meaning in Java

Q5. Explain class variable briefly.

Ans. Class variables are declared inside the class as its member with `static` modifier. Class variables are associated with class and allocate only one memory space for all the objects of that class. In other words, same value of the class variable is shared by all the objects of that class. Class variables are available in the program till the existence of their associated class.

Q6. Explain inner classes briefly.

Ans. A normal nested class declared without using the static keyword is called non-static nested class. It is also called inner class. The non-static nested classes can access the instance variables of the inclosing classes directly, without instating the enclosing class. Therefore, you can conclude that the non-static nested class is fully within the scope of the enclosing class.

Q7. Explain call by reference briefly.

Ans. When you pass the object reference to the methods as its parameter, such method invocations are called, call by reference. In call by reference, objects are passed as the method parameter. Any change made to the dummy parameter is also reflected in the actual parameter. This is because; both the actual and dummy parameters refer to the same object.

Q8. Explain method overloading briefly.

Ans. Method overloading is an object-oriented technique that lets you define several different versions of a method with the same name, but each with a different parameter list. When you use an overloaded method, the Java compiler will know which one you mean by the number and/or types of the parameters you pass to the method, finding the version of the method with the right parameter list.

Q9. Explain method overriding briefly.

Ans. Method overriding is also an object-oriented technique that lets you define methods with the same name in the super class and subclass when you implement inheritance. When you use method overriding, the Java compiler decides which method to be invoked by the object, which is used to invoke that method.

Q10. What are constructors? What are the points to remember while working with constructors?

Ans. Constructors are special type of methods having the same name as the class name, for example, if a class carries a name Demo then the constructor will also have the name Demo. Constructors are syntactically similar to methods and used to initialize the objects. A constructor initializes the object as soon as the object is created. Following are few important points that must be remembered while working with constructors:

- The name of the constructor should be same as that of a class.
- A constructor does not have any return type—not even void—because implicitly, the return type of a constructor is a class itself.
- First line of a constructor must either be a call to other constructor in the same class or a call to super class constructor. If you do not call other constructor or super class constructor explicitly, then the compiler implicitly calls the parameter less constructor of its super class.

Q11. Explain automatic typecasting briefly.

Ans. In the automatic type casting, it is necessary that the two data types being used must be compatible to each other. In Java, the numeric data types such as integers and floats are compatible to each other. These data types are not compatible with boolean and char data types. For automatic type casting, another necessary fact is that the destination data type must be longer than the source data type. When these two conditions meet then only the automatic type conversion takes place.

Q12. Explain for-each loop briefly.

Ans. The for-each loop is specifically designed to handle the elements of a collection. A collection is represented by a group of elements. The for-each loop repeatedly executes a group of statements for each element of the collection. It executes as many times as there are number of elements in the collection.

Debugging Exercises

Q1. The following program results in compile time error. Identify the error.

```
class Question1 {
 public static void main(String ar[]) {
     short b1;
     int b2=3;
     b1=b2;
     System.out.println("Result= "+b1);
 }
}
```

Ans. Preceding program gives compile time error in line 5 shown as follows:

```
b1=b2;
```

In this assignment, an int value is being assigned into a short variable, which is illegal. This can be achieved by explicit typecasting as shown in the following program:

```
class Question1 {
 public static void main(String ar[]) {
     short b1;
     int b2=3;
     b1=(short)b2;
     System.out.println("Result= "+b1);
 }
}
```

Q2. What modification should be done in the following program to avoid the compile time error?

```
class Question2 {
 public static void main(String ar[]) {
     float a=10.5;
     float b=a;
     System.out.println("Result= "+b);
 }
}
```

Ans. Preceding program gives compile time error in line 4 shown as follows:

```
float a=10.5;
```

In this line, a float variable is declared and initialized without using the postfix f. As in Java, all the floating-point numbers are considered as double by default. The compile time error can be avoided by adding f at the end of 10.5, as shown here:

```
float a=10.5f;
```

By adding f at the end of 10.5, the compiler is intimated that the value being assigned is a float value.

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

```
class Question3 {
 int x;
 void display() {
     System.out.println(x);
 }
 public static void main(String ar[]) {
     Question3 Q=new Question3();
     Q.display();
 }
}
```

Ans. Output of the preceding program is 0. In this program, the x instance variable is not initialized with any value, therefore it is initialized with its default value 0. This default value 0 is displayed as output.

Q4. Identify the error in the following program:

```
class Question4 {
 public static void main(String ar[]) {
System.out.println(x);
 }
}
```

Ans. In the preceding program, the compilation error occurs in line number 3, as shown here:

```
System.out.println(x);
```

The reason of this compilation error is the use of x variable without its declaration and initialization. To make the program error free, you must declare and initialize the x variable as shown in following program:

```
class Question4 {
 public static void main(String ar[]) {
     int x=5;
     System.out.println(x);
 }
}
```

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

```
class Question5 {
 public static void main(String[] args) {
     int mark=10;
     if(mark<10) {
         System.out.println("You are fail");
     }
 }
}
```

Ans. When the preceding program is compiled and executed, nothing is displayed as output. This is because, the mark variable is initialized with 10 and the boolean expression for if checks whether it is less than 10. This condition is false and therefore, the statement inside the if block is not executed.

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

```
class Question6 {
 public static void main(String[] args) {
     int mark=10;
     if(mark<10) {
         System.out.println("You are fail");
     }
     else
     System.out.println("You are pass");
 }
}
```

Ans. When the preceding program is compiled and executed, You are pass is displayed as output. This is because, the mark variable is initialized with 10 and the boolean expression for if checks whether it is less than 10. This condition is false and therefore, the statement inside the if block is not executed, rather the statement inside the else block is executed.

Q7. What modification must be done in the following program to avoid compilation error?

```
class Question7 {
 int roll=1;
 public static void main(String[] args) {
     obj=new Question7();
     System.out.println(obj.roll);
 }
}
```

Ans. When the preceding program is compiled, an error occurs in line number 4, as shown here:

```
obj=new Question7();
```

This error occurs because the obj reference is not declared and being used for holding an object. You need to declare the obj reference variable of the Question7 class before using it. The modified program is shown as follows:

```
class Question7 {
 int roll=1;
```

```
    public static void main(String[] args) {
        Question7 obj;
        obj=new Question7();
        System.out.println(obj.roll);
    }
}
```

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

```
public class Question8 {
 public static void main(String args[]) {
     char country='i';
     switch(country) {
         case 'b': System.out.println("you are in Brazil");
         break;
         case 'n': System.out.println("You are in Newzealand");
         break;
         case 'i': System.out.println("You are in India");
         break;
         default: System.out.println("You are no where");
     }
 }
}
```

Ans. When you compile and execute the preceding program, it will show "You are in India" as output.

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

```
class Question9 {
 public static void main(String args[]) {
     int i=10;
     do {
         System.out.print(i+" ");
         i++;
     }while(i<15);
 }
}
```

Ans. When you compile and execute the preceding program, it will show 10 11 12 13 14 as output.

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

```
class Question10 {
 public static void main(String args[]) {
     int i;
     for(i=1; i<5; i++) {
         System.out.print(i+" ");
     }
 }
}
```

Ans. When you compile and execute the preceding program, it will show 1 2 3 4 as output.

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

```
class Question11 {
 public static void main(String[] args) {
     int j = 0;
     while (true){
         j=j+2;
         if (j==10) {
             break;
             }
         System.out.print(j+" ");
     }
 }
}
```

Ans. When you compile and execute the preceding program, it will show 2 4 6 8 as output.

Q12. What will be output of the following program?

```
class Question12 {
 public static void main(String[] args) {
     for (int i=2;i<=10;i=i+2) {
         if (i == 6) {
             continue;
         }
         System.out.print(i+" ");
     }
 }
}
```

Ans. When you compile and execute the preceding program, it will show 2 4 8 10 as output.

Q13. What will be output of the following program?

```
class Question13 {
    public static void main(String args[]) {
        int NO=25;
        int R,SUM=0,F=1;
        do {
            R=NO%2;
            SUM+=R*F;
            NO=NO/2;
            F*=10;
        }
        while(NO!=0);
        System.out.println("Sum="+SUM);
    }
}
```

Ans. When you compile and execute the preceding program, it will show "Sum=11001" as output.

Q14. What will be output of the following program?

```
class Question14 {
    public static void main(String args[]) {
```

```
            int NO=81;
            double K=Math.sqrt(NO);
            if (NO==K*K)
            System.out.println("The NO is perfect Sqrt");
            else System.out.println("the NO is not a perfect Sqrt");
        }
    }
```

Ans. When you compile and execute the preceding program, it will show "The NO is perfect Sqrt" as output.

Q15. What will be output of the following program?

```
import java.lang.Math;
class Question15 {
    public static void main(String args[]) {
        int I,K,J,T;
        for(I=1;I<=40;I++)
        for(J=I;J<=40;J++)     {
            T=I*I+J*J;
            K=(int)Math.sqrt(T);
            if(K*K==T)
            System.out.print(I+J+K + " ");
        }
    }
}
```

Ans. When you compile and execute the preceding program, it will show "12 30 24 56 40 36 90 60 48 84 60 90 80 72 70 84 96 108 120" as output.

Q16. What will be output of the following program?

```
class Question16 {
    public static void main(String args [])  {
        int I;
        for (I=100;I<=9999;I++) {
            int NO=I,SUM=0,d;
            do
            {
                d=NO%10;
                SUM=SUM+d*d*d;
                NO=NO/10;
            }
            while(NO>0);
            if (SUM==I)
            System.out.print(I + " ");
        }
    }
}
```

Ans. When you compile and execute the preceding program, it will show "153 370 371 407" as output.

Q17. What will be output of the following program?

```
class Question17 {
    public static void main(String args [])  {
        int I=2;
```

```
            for (I=2;I<=100;I=I+2)
            if ((I%6!=0)&&(I%8!=0))
            System.out.print(" "+I);
        }
    }
```

Ans. When you compile and execute the preceding program, it will show "2 4 10 14 20 22 26 28 34 38 44 46 50 52 58 62 68 70 74 76 82 86 92 94 98 100" as output.

Q18. What will be output of the following program?

```
class Question18 {
 public static void main(String args[]) {
     int i=1,cnt=1;
     System.out.println(" "+i);
     for(int a=1;a<=9;a++)
     {
         i=i*10;
         ++cnt;
     System.out.println(" "+i);
     }
 }
}
```

Ans. When you compile and execute the preceding program, it will show

```
1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
```

as output.