Table of Contents

Introduction ..... *xv*

Chapter 1: Java Fundamentals 1

1.1 Introducing Object Oriented Programming 1

Explaining OOP Concepts 2

1.2 Evolution of Java 5

Emergence of C++ 5

Introducing Java 6

1.3 Comparing Java with Other Programming Languages 8

1.4 Features of Java 9

Simple 10

Secure 10

Portable 10

Robust 10

Multithreading 11

Platform Independent 11

Distributed 11

Dynamic 11

1.5 Exploring New Features of J2SE 5.0 12

Generics 12

Enhanced for Loop 12

Autoboxing and Unboxing 12

Typesafe Enums 13

Varargs 13

1.6 Introducing the Java Environment 13
Explaining JDK 13
Explaining Java Platform Components 14
Types of Java Programs 16
1.7 Developing a Simple Java Program 16
Writing the Java Program 17
Saving the Java Program 19
Compiling the Java Program 20
Executing the Java Program 21
1.8 Working with Java Tokens 21
Keywords 22
Identifiers 23
Using Literals 24
Explaining Constants 32
Explaining Separators 33
Using Operators in Java 34
Summary 54
Review Questions 55
True or False 55
Multiple Choice Questions 55
Short Answer Questions 57
Debugging Exercises 59

Chapter 2: Working with Java Members and Flow Control Statements 63

2.1 Explaining Data Types 64
Integer Data Types 64
Floating-point Data Types 65

Character Data Type 66
The boolean Data Type 66
2.2 Declaring Variables 67
Declaring the Instance Variables 68
Declaring the Class Variables 68
Declaring the Local Variables 69
2.3 Declaring Classes 70
Declaring Nested Classes 72
2.4 Declaring Methods 73
Methods Returning a Value 75
Passing Arguments to Methods 77
Overloading and Overriding of Methods 80
2.5 Explaining Constructors 83
Defining Simple Constructors 83
Defining Parameterized Constructors 84
Overloading Constructors 86
2.6 Creating Objects 87
2.7 Explaining Access Specifiers 88
2.8 Explaining Type Casting and Type Conversion 89
Automatic Type Casting 89
Casting Incompatible Data Types 90
2.9 Implementing Flow Control Statements 91
Using Selection Statements 91
Using Iteration Statements 101
Using Jump Statements 128
Summary 133
Review Questions 133

True or False 134
Multiple Choice Questions 134
Short Answer Questions 138
Debugging Exercises 140

Chapter 3: Working with Arrays, Vectors, Strings, and Wrapper Classes ... 147

3.1 Introduction to Computers 147
3.2 Using Arrays in Java 148
Using One-Dimensional Array 148
Using Multi-dimensional Array 161
Using Array of Objects 173
Using the System.arraycopy() Method 174
3.3 Using Vectors in Java 176
Defining the Enumeration Interface 176
Defining the Vector Class 177
Demonstrating Vector in a Program 180
3.4 Using the Wrapper Classes in Java 182
Using the Integer Wrapper Class 184
Using the Double Wrapper Class 184
Using the Character Wrapper Class 185
Using the Boolean Wrapper Class 186
Using Methods of the Wrapper Classes 186
3.5 Using Strings in Java 188
Defining the String Class 189
Defining Arrays of the String Types 191
Implementing Strings Concatenation 193
Using Methods of the String Class 195

Implementing the charAt() Method 201
Comparing the equals() Method and == Operator 215
Using the StringBuffer Class 216
Summary 219
Review Questions 220
True or False 220
Multiple Choice Questions 220
Short Answer Questions 222
Debugging Exercises 225

Chapter 4: Exception Handling and I/O Operations 229

4.1 Handling Exceptions 229
Errors and Exceptions in a Java Program 231
Exception Handling 234
Handling Multiple Exceptions 245
Types of Exceptions 248
4.2 Handling I/O Operations 250
Reading Console Input 251
Reading Characters 252
Reading Strings 253
Writing Console Output 256
Reading and Writing Files 265
Summary 269
Review Questions 269
True or False 269
Multiple Choice Questions 270

Short Answer Questions 271
Debugging Exercises 273

Chapter 5: Implementing Inheritance in Java 279

5.1 Understanding Inheritance 279
5.2 Using Forms of Inheritance 281
Implementing Single Inheritance 281
Implementing Hierarchical Inheritance 284
Implementing Multilevel Inheritance 287
5.3 Identifying Inheritance and Member Accessibility 289
Using the public Access Modifier 289
Using the protected Access Modifier 291
Using the private Access Modifier 293
5.4 Referencing Subclass Objects 295
5.5 Using the super Keyword 297
5.6 Invocation of Constructors in Inheritance 299
5.7 Using the final Keyword 301
5.8 Declaring abstract Classes 303
5.9 Working with Interfaces in Java 306
Defining Interfaces 306
Implementing Interfaces 307
Using Reference Variables of an Interface 309
Extending Interfaces 310
Accessing Interface Variables 312
5.10 Exploring Class Hierarchy in Java 314
Summary 315
Review Questions 315

True or False 315
Multiple Choice Questions 316
Short Answer Questions 318
Debugging Exercises 320

Chapter 6: Multithreading and Packages in Java 325

6.1 Overview of Threads 325
6.2 Defining a Thread 326
The Main Thread 327
Extending the java.lang.Thread 327
Implementing the java.lang.Runnable Interface 328
6.3 Instantiating a Thread 328
6.4 Starting a Thread 330
Starting and Running Multiple Threads 332
6.5 Thread States and Transitions 334
Thread States 334
Thread Priorities 336
Thread Scheduler 338
Running and Yielding 338
Sleeping and Waking Up 339
Joining 341
Blocking for I/O 345
Thread Termination 345
Deadlocks 345
6.6 Code Synchronization 346
Locks 346
Synchronized Methods 347

Synchronized Blocks 350
6.7 Thread Interaction 351
6.8 Working with Packages in Java 355
Creating a Package 355
Importing Packages 358
Demonstrating Access Protection 360
6.9 Defining Java API Packages 363
The java.lang Package 363
The java.util Package 372
Summary 381
Review Questions 382
True/False Questions 382
Multiple Choice Questions 382
Short Answer Questions 384
Debugging Exercises 386

Chapter 7: Working with Applets 393

7.1 Overview of Applets 394
7.2 Life Cycle of an Applet 394
7.3 Comparing Applets and Applications 396
7.4 Creating Applets 397
Using the HTML Applet Tag 400
Initializing and Terminating an Applet 405
Using the paint() Method 406
Using the update() Method 406
Invoking the repaint() Method 407
Implementing Multithreading with Applets 408

Passing Parameters to Applets 413
7.3 Working with the Graphics Class 415
Drawing Lines 419
Drawing Rectangles 419
Drawing Ovals 420
Drawing Arcs 421
Drawing Polygon 422
7.6 Working with the Color Class 423
7.7 Working with the Font Class 427
Determining the Available Fonts 431
Creating and Selecting Fonts 432
Using the FontMetrics Class 434
7.8 Handling Events 437
The Delegation Event Model 437
Events Source 438
Explaining Event Listeners 438
Implementing the Delegation Event Model 448
Using Adapter Classes 451
Using Anonymous Inner Adapter Classes 452
Summary 453
Review Questions 453
True False 454
Multiple Choice Questions 454
Short Answer Questions 456
Debugging Exercises 459

Chapter 8: Window-Based Applications in Java 465

8.1 Introducing AWT 465

Exploring the Component Class 466

Exploring the Container Class 467

Exploring the Panel Class 468

Exploring the Window Class 468

Exploring the Frame Class 468

8.2 Using AWT Frames 468

Creating a Frame Window 469

Hiding and Showing the Frame Window 470

Setting the Title of the Frame Window 471

Closing a Frame Window 473

8.3 Handling Events in Frames 475

8.4 Using AWT Components 477

Handling Push Buttons 478

Adding Labels and Text Fields 481

Adding Check Boxes and Radio Buttons 483

Handling a Choice List 486

Handling Lists 488

Adding a Scrollbar 491

Adding a Text Area 493

8.5 Using Layout Managers 495

The FlowLayout Manager 495

The BorderLayout Manager 497

The GridLayout Manager 500

The GridBagLayout Manager 502

The CardLayout Manager 507

8.6 Working with Menus and Menu Bar 510
8.7 Working with Dialog Boxes 515
8.8 Working with Images 520
Summary 523
Review Questions 523
True or False 523
Multiple Choice Questions 524
Short Answers 525
Debugging Exercises 527
Additional Questions 529
Engineering Question Paper 539
Solutions for Engineering Questions Paper 545
Index 563
What's on the CD-ROM 568

```
Command Prompt
D:\code\chapter3>java VectorExample
Initial Capacity of the created Vector is: 5

Capacity of the vector after increment capacity is: 11

Size of the vector is: 9

First item in the vector is: 0

Last item in the vector is: 3.14157

Yes! integer value 3 is available.

Element objects present in this vector are :
0
1
2
3
4
5
6
3.14159
3.14157

D:\code\chapter3>
```

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

In this book, tables are placed immediately after their first references, as follows:

Table 1 describes the timeline of Java:

Table 1: Timeline Showing Development Stages of Java

Year	Development stages
1990	In 1990, Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named Oak in 1991.
1992	The team, known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with tiny touch-sensitive screen.
1993	The World Wide Web (WWW) appeared on the Internet and transformed the text-based Internet into a graphical-rich environment. The Green Project team came up with the idea of developing Web applets using the new language that could run on all types of computers connected to Internet.

Table 1: Timeline Showing Development Stages of Java

Year	Development stages
1994	The team developed a Web browser called HotJava to locate and run applet programs on Internet. HotJava demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed as Java. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as a leader for Internet programming but also as a general-purpose, OOP language. Sun releases Java Development Kit 1.0 (JDK 1.0).
1997	Sun releases JDK 1.1.
1998	Sun releases the Java 2 with version 1.2 of the Software Development Kit (SDK 1.2).
1999	Sun releases the Java 2 Platform, Standard Edition (J2SE), and Enterprise Edition (J2EE).
2000	Sun releases J2SE with SDK 1.3.
2002	Sun releases J2SE with SDK 1.4.
2004	Sun releases J2SE with JDK 5.0.
2006	Sun releases J2SE with JDK 6.0.

You can also find some problem solving questions, related to the explained topics, with their solutions inside the *Test Your Knowledge* section. The format of a problem solving question is given as follows:

TEST YOUR KNOWLEDGE

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

Ans.

```
import java.util.*;
class TestYourKnowledge25
{
    public static void main(String args[])
    {
        Vector vect=new Vector(5,3);
        vect.addElement(new Integer(254));
        vect.addElement(new Double(214.25));
        vect.addElement(new Integer(142));
        vect.addElement(new Boolean(false));
```

01 Java Fundamentals

IN THIS CHAPTER

1.1 Introducing Object Oriented Programming
1.2 Evolution of Java
1.3 Comparing Java with Other Programming Languages
1.4 Features of Java
1.5 Exploring New Features of J2SE 5.0
1.6 Introducing the Java Environment
1.7 Developing a Simple Java Program
1.8 Working with Java Tokens

Java is an Object Oriented Programming language that inherits its features from C++. Java language was created by a software developer, James Gosling, at Sun Microsystems in 1991. It was first called Oak; however, the name got changed to Java in 1995, when it was first released for public use.

Before the advent of Java and C++, C was an extremely popular language among programmers, and it seemed that C was the perfect programming language combining the best elements of low-level language and higher-level languages into a programming language. However, C had its limitations; for example, when C programs grew longer, they became more unwieldy, because programmers could not reduce long C program into self-contained compartments. This meant that the code in the first line of a long program could interfere with the code in the last line, and the programmer had to keep the whole code in mind while programming. Then, the concept of Object Oriented Programming (OOP) was introduced to cut long programs into semi-autonomous units. This lead to the emergence of C++ programming language.

The chapter begins with the discussion on OOP and also discusses the need and consequent evolution of Java. Moreover, the chapter also helps in comparing Java with other programming languages such as C and C++ and states the features of Java. The Java environment comprising of Java Development Kit (JDK), Java Platform components, and types of programs, which are also discussed. Towards the end, a simple Java program is developed to provide a flavor of Java programming and then the Java tokens, such as identifiers, keywords, literals, and operators are discussed

1.1 Introducing Object Oriented Programming

OOP is a technique that allows you to implement that famous programming dictum: "divide and conquer". The idea behind OOP is to encapsulate private data and methods into objects, thereby making each object semi-autonomous.

```
        vect.addElement(new Integer(4));
        Enumeration e= vect.elements();
        System.out.println("Element objects present in this vector
        are : "  );
        while(e.hasMoreElements())
        {
                System.out.println(e.nextElement());
        }
        System.out.println("Size of the dynamic array is: " +
        vect.size());
    }
}
```

Execution:

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

Output:

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

The concept of OOP can be understood better with the help of a scenario. Imagine you have placed several pipes, pumps, a compressor, and all kinds of switches in your kitchen to keep food cool. Every time the temperature of the food gets too high, you turn on the compressor, open the valves, and start cranking the pumps manually. Now, suppose you wrap all these functionalities into an object, say, a refrigerator, in which all these operations are handled automatically inside the object with proper coordination between the various parts. This is the concept behind encapsulation; or in other words, encapsulation hides the complexity of a class by providing an object of a class. The object handles all the work internally and can be easily conceptualized, similar to the preceding example of refrigerator. If the first dictum of OOP is "divide and conquer", the second is surely "out of sight, out of mind". Now, let's discuss some fundamental concepts in OOP.

Explaining OOP Concepts

The OOP provides basic concepts which help in programming. Java, being an Object Oriented Programming language, implements and applies the concept of OOP. Consequently, you must become familiar with the OOP concepts. The basic concepts of OOP are as follows:

- Objects
- Classes
- Data Hiding and Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Now let's discuss each of them in detail.

Objects

`Objects` are basic runtime entities in OOP. An object can represent a person, a place, a bank account, or user-defined data such as vectors, lists, and collections. When a program is executed, the objects interact with each other by sending messages. For example, if employee and salary are two objects in a company program, the employee object may send a message to the salary object requesting the annual salary. Each object contains data, and code to manipulate that data.

OOP allows decomposing a problem into a number of entities known as objects and then the methods are invoked using the objects. Therefore, an object is a combination of data and methods.

Classes

In OOP, `classes` provide a sort of `template` for `objects`. You use a class to create an object and then you can call the object's methods from your code. In other words, objects are variables of type class. After defining a class, any number of objects belonging to that class can be declared.

A class is a collection of objects of similar type. For example, car, bike, and scooter are members of the vehicle class. The syntax for declaring an object of a class is:

```
Class_name Object_name;
```

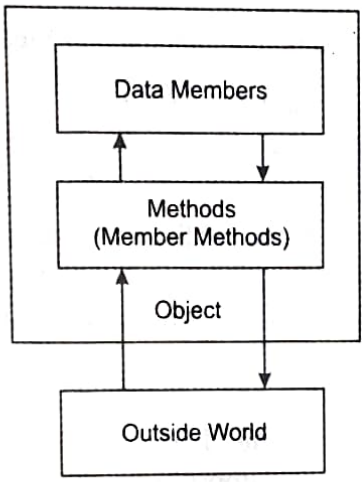
Now considering the example of the vehicle class, the following code snippet declares the object of the vehicle class:

```
vehicle v;
```

In the preceding code snippet, the v object is declared of the vehicle class.

Data Hiding and Encapsulation

The concept of wrapping data and functions into a single unit is known as encapsulation. It is a mechanism that binds class members (variables and methods) together and prevents them from being accessed by other classes. The term data hiding implies that the data is not accessible by other classes and only the methods which are wrapped within the class can access it. The methods that serve as a mediator between the object's data and the program are known as member methods. Figure 1 shows the concept of data hiding:



▲ **Figure 1: Displaying Data Hiding**

In Figure 1, the data members are hidden from the other classes and thereby they can only access the member methods.

Data Abstraction

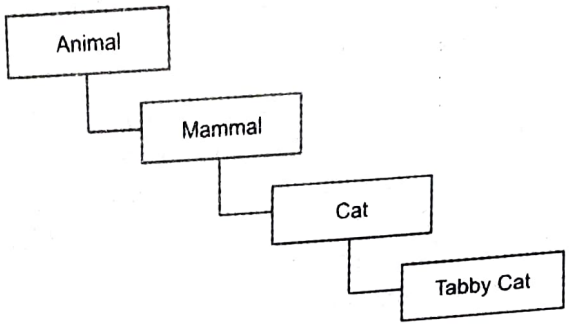
The concept of representing important features without including the background details is known as `abstraction`. Abstraction is the mechanism by which we hide data that is not required by a user. The advantage of abstraction is that the user can work only with the required data and does not need to view unwanted data.

Let's take the example of the car to understand the concept of abstraction. As the owner of the car, you must know how to drive it. This obviously implies that you know about the various components of the car and how to use them. For example, you know that the accelerator pedal is used to increase the speed of the car and pressing the brake pedal stops it. To perform these simple actions, you just need to know how to use these components, and not how they function. Similarly, in OOP, you need to know about the specific classes or methods that are called to implement specific program logic, without knowing how these classes or methods work. This concept is known as abstraction.

Inheritance

Inheritance is one of the OOP concepts by which a class acquires the properties of another class. A class that is inherited from another class is known as the subclass and the class from

which the subclass is inherited is known as the superclass or the base class. The concept of inheritance supports hierarchical classification. Without inheritance, you need to define all the characteristics of each object explicitly. For example, a `TabbyCat` is a type of cat, which, in turn, belongs to the mammal class, which comes under the larger class, animal. Figure 2 shows the hierarchical tree of this example:



▲ **Figure 2: Displaying the Hierarchical Tree Demonstrating Inheritance**

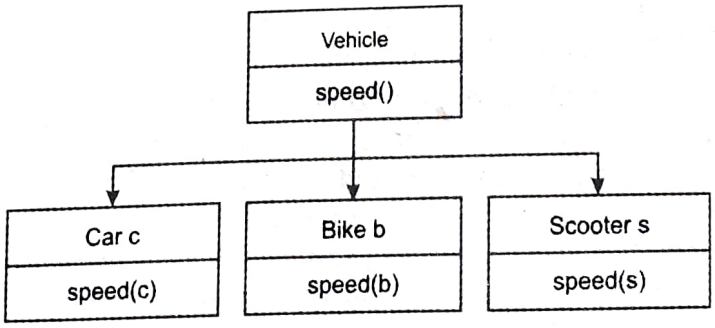
Now, let's look at Figure 2 from the programming aspect. Here, the `Animal` class is a `superclass` and Mammal is the subclass extending or inheriting all the attributes of the `superclass`.

In OOP, the inheritance concept facilitates reusability as we can append additional features in an existing class without modifying it by deriving a new class from the existing one. The subclass derived from the `superclass` contains the combined features of both the classes.

Polymorphism

The term `polymorphism` comes from two Greek words, poly, meaning many, and morphs, meaning forms. Therefore, polymorphism means many forms. In OOP, polymorphism allows you to perform various operations by using the same method. The behavior of an operation depends upon the types of data used in the operation.

For example, let's consider the operation of addition, which can be addition of two numbers or concatenation of two strings. In the first case, the operands are of int type, and in the other case, the operands are strings. The concept of using a single method name to perform different types of tasks is known as method overloading. Figure 3 shows that a single method, `speed()`, is used to handle different types of arguments:



▲ **Figure 3: Displaying Polymorphism**

In Figure 3, the `speed()` method of the `Vehicle` class is used in different forms in the `Car`, `Bike`, and `Scooter` classes. The objects of the respective class is passed as an argument to the `speed()` method. Moreover, it is important to note that polymorphism is extensively used in implementing inheritance.

➡ Dynamic Binding

The process of linking a method call to the executable code in response to the call is known as `binding`. The binding process exhibited at runtime is called dynamic binding. This means that when a method is called, the method call is bound to the method body during the runtime of the program. The Java compiler is not aware of the method to be invoked during compilation; therefore, the `JVM` invokes the relevant method during runtime. Hence, this is also called runtime polymorphism or dynamic binding.

➡ Message Passing

An object-oriented program comprises a set of objects that communicate with each other. The following steps are involved in the process of programming in OOP:

- Create classes that define objects and their behaviors
- Create objects from class definitions
- Establish communication among objects

The one or more objects can communicate with one another by sending and receiving information. This process is similar to the way people pass messages to each other. A message for an object is a request for execution of a method. Then, the method is invoked which provides the desired result. Message passing involves specifying the name of the object, the name of the method, and the information to be sent. The following code snippet shows message passing:

```
Car c;
c.speed(c);
```

In the preceding code snippet, the `c` instance of the `Car` class invokes the `speed()` method and `c` is passed as an argument.

Now, after understanding the OOP concepts, let's discuss the evolution of Java.

1.2 Evolution of Java

Java has inherited some features from C and C++ programming languages, such as it derives its syntax from C and many of its object oriented features are influenced by C++. Here, we discuss the sequence of events and forcing factors that led to the evolution of Java.

During the late 1970s and early 1980s, C became the most important and widely used computer programming language. Although C gained popularity, but due to the increasing complexity in its programs, programmers started looking for better options. During this time, OOP became popular as it was used to manage and cut complex programs into semi-autonomous units.

■ Emergence of C++

When OOP concept was added to C, it became C++, and programmers had a new programming language to work with. C++ allows programmers to deal with complex programs, and object-oriented code which helped to solve many other problems as well. For example, the support of objects in C++ made it easier for the manufacturers to provide the user with lots of prewritten code in the form of classes. You can simply create an object to use a *class*, which acts like a template for that object.

C++ was developed by Bjarne Stroustrup at AT & T Bell Laboratories in Murray Hill, New Jersey, USA, in early 1980s. C++ is an extension of C with major addition of the class construct feature of Simula67. Since the major additional feature with respect to C was class, Stroustrup initially called the new language as C with classes. However, later in 1983, the name changed to C++.

The support of classes in C++ enabled the software manufacturers to provide you with numerous libraries containing classes, so that you can create objects of these classes. For example, one of the most popular libraries of C++ classes is the `Microsoft Foundation Class` (MFC) library that comes with Microsoft's Visual C++. Earlier, to write a Windows program in C, you needed about five pages of solid code just to display a blank window. However, using a class in the MFC library, you could simply create an object of the kind of window you want to use, such as with a border, without a border, as a dialog box. The object already had built-in functionality of the kind of window you wanted to create, so only one line code to create the new window object is provided.

Even more impressive was the fact that you could use an MFC class as a *base class* for your own classes. This would enable you to add any functionality to that class through a process called inheritance in OOP. For example, suppose you want your window to display a menu bar, you can derive your own class from a plain MFC window and add a menu bar to that class. In this way, you can build your own class just by adding a few lines of code for adding a menu bar.

All this seemed great to programmers, and C++ became popular among programmers. Most of the programmers felt that C++ is the perfect programming language and no other programming language could be better than C++. However, the programming environment itself was about to undergo a great change with the popularization of an immense new programming environment—the Internet and that's what made Java so popular.

Introducing Java

The first version of Java was introduced in 1991 and was called Oak, instead of Java. The original idea for Oak was to create a platform-independent, object-oriented programming language. The driving inspiration behind Oak was to create something that could be used on all the computers. Now that Java has been popularized by the Internet, a huge and increasing number of corporations are adopting it internally instead of C++ for just that reason—in fact, some versions of C++, such as Microsoft's C#, are now being modeled after Java. Oak was renamed Java in 1995, when it was released for public use. By that time, Java had adopted a model that made it perfect for the Internet—the bytecode model.

A Microsoft Visual C++ program is large, typically starting off at a minimum of 5 MB for a full MFC program, and that doesn't even count the Dynamic Link Libraries (DLLs) that the Windows platform needs to run Visual C++ programs. Due to large size, it becomes difficult to download all that as part of a Web page to perform some specified task on your computer.

Java programs, on the other hand, are constructed differently. Java itself is implemented as the JVM, which is the application that actually runs your Java program. When JVM is installed on a computer, it can run Java programs. Java programs, therefore, don't need to be self-sufficient, and they don't have to include all the machine-level code that actually runs on the computer. Instead, Java programs are compiled into compact bytecodes, and it's these bytecodes that the JVM reads and interprets to run your program. When you download a Java applet on the Internet, you are actually downloading a bytecode file.

In this way, your Java program can be very small, because all the machine-level code to run your program is already on the target computer and doesn't have to be downloaded. To host Java on all computers, Sun only had to rewrite JVM to work on those computers. As your program is stored in a bytecode file, it will run on any computer on which JVM is installed.

Java programs were originally interpreted by the JVM, which performed bytecode by bytecode execution. This made interpretation a slow process. To overcome the slow process, Java 2 introduced the Just In Time (JIT) compiler, which is built into the JVM. The JIT compiler actually reads your bytecodes in sections and compiles them interactively into machine language so the program can run faster (the whole Java program is not compiled at once because Java performs runtime checks on various sections of the code).

Now after understanding the emergence of Java, let's now discuss the development stages of Java.

History of Java

Java is a simple, portable, and highly reliable programming language. Here, we discuss the development stages of Java. Table 1 describes the timeline of Java:

Table 1: Timeline Showing Development Stages of Java

Year	Development stages
1990	In 1990, Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named Oak in 1991.
1992	The team, known as Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with tiny touch-sensitive screen.
1993	The World Wide Web (WWW) appeared on the Internet and transformed the text-based Internet into a graphical-rich environment. The Green Project team came up with the idea of developing Web applets using the new language that could run on all types of computers connected to Internet.
1994	The team developed a Web browser called HotJava to locate and run applet programs on Internet. HotJava demonstrated the power of the new language, thus making it instantly popular among the Internet users.
1995	Oak was renamed as Java. Many popular companies including Netscape and Microsoft announced their support to Java.
1996	Java established itself not only as a leader for Internet programming but also as a general-purpose, OOP language. Sun releases Java Development Kit 1.0 (JDK 1.0).
1997	Sun releases JDK 1.1.
1998	Sun releases the Java 2 with version 1.2 of the Software Development Kit (SDK 1.2).
1999	Sun releases the Java 2 Platform, Standard Edition (J2SE), and Enterprise Edition (J2EE).

Table 1: Timeline Showing Development Stages of Java

Year	Development stages
2000	Sun releases J2SE with SDK 1.3.
2002	Sun releases J2SE with SDK 1.4.
2004	Sun releases J2SE with JDK 5.0.
2006	Sun releases J2SE with JDK 6.0.

Let's now discuss the comparison of Java with other programming languages, such as C and C++.

1.3 Comparing Java with Other Programming Languages

Here, we discuss the comparison of Java with other languages, such as C and C++. Let's first compare it with C language. Java inherits the coding syntax of C programming language; however, there are many differences between C and Java programming languages. Table 2 compares the Java and C programming languages:

Table 2: Comparison between Java and C

Points of Difference	Java	C
Type of programming language	It is an object-oriented programming language.	It is a structured programming language.
Support for struct and union data type	It does not provide support for the struct and union data type.	It provides the support for struct and union data type.
Support for pointer	It does not provide support for an explicit pointer type.	It supports the concept of pointers.
Preprocessor	It does not have a preprocessor and therefore, it does not provide support for the #define, #include, and #ifdef statements.	It has the preprocessor directives.
Methods without arguments	It declares methods with no arguments with empty parenthesis and not with the void keyword as done in C.	It uses the void keyword to specify that the method is without argument.
New Operators	It provides new operators, such as instanceof and >>>.	It does not provide support for the `instanceof` and >>> operators.
Support for break and continue statements	It provides support for labeled break and continue statements.	It uses simple break and continue statements.

Now after comparing Java with C programming language, let's discuss the comparison of Java with C++.

Java has adopted object oriented features from C++ language. Table 3 compares Java with C++:

Table 3: Comparison between Java and C++

Points of Difference	Java	C++
Support for operator overloading	It does not support operator overloading.	It supports operator overloading.
Concept of template classes	It does not provide the concept of template classes.	It provides support for template classes.
Support for multiple inheritance	It does not provide support for multiple inheritance of classes, rather it is accomplished by a new feature called interfaces.	It provides support for multiple inheritance.
Support for global variables	It does not support global variables. Every variable and method is declared as a class member.	It provides support for global variables.
Support for pointers	It does not provide support for pointers.	It uses the concept of pointers.
Destruction of Objects	Similar to C++, Java does not use destructor function instead invoke the finalize() method to destroy objects.	It uses the destructor function to destroy objects.
Concept of header files	It does not have any header file.	You need to include header files in C++.
Concept of multithreading	It provides support for multithreading.	It does not have multithreading support.
Garbage Collection	It supports automatic garbage collection.	It does not have automatic garbage collection.
Keywords	It does not include the following keyword of C++: ▪ delete ▪ friend ▪ inline ▪ mutable ▪ template ▪ using ▪ virtual	The following keywords are excluded by Java, but considered as keywords in C++: ▪ #define ▪ const

Now, after discussing this comparison of Java with C and C++, let's discuss the general features and benefits of Java.

1.4 Features of Java

Java is a simple, robust, platform independent, and portable programming language. It is designed to meet the real world requirements with its following features:

- Simple

- Secure
- Portable
- Robust
- Multithreaded
- Platform Independent
- Distributed
- Dynamic

Simple

Java is simple to learn and use. In Java, it is easy to write and debug programs because Java does not use pointers. Moreover, Java provides bug free system due to strong memory management and also has automatic memory allocation and deallocation system. Unlike other programming languages, which provide dozens of complicated ways to perform a simple task, Java provides a simple and clear approach to achieve a given task. If a programmer understands the basic concepts of OOP, it becomes easier for him to learn Java.

Secure

In today's world, everybody needs safety and security. Due to the threat of hackers, people feel unsafe while performing commerce over Internet. In order to overcome such fears, Java provides the safety and security features.

While using Java compatible browser, everyone can safely download Java applets by relaxing their minds from the fear of virtual infection or malicious intent. Since the Java programs are confined to the Java runtime environment, so they can access only that part of your computer which is required for program execution. The data outside Java runtime environment cannot be accessed by Java programs, and therefore downloading a Java application through Internet will not harm your computer. That's why Java is a secure language.

Portable

In the distributed world of Internet, an application developed with the help of a programming language might be accessed on various computers having different kinds of operating systems. But, it's not guaranteed that the application is portable, that is the application runs successfully on all operating systems. To overcome this portability issue, Java introduces a concept of what is known as bytecode.

Bytecode is a set of instructions that is generated by Java compiler while compiling a Java program. In other programming languages, a program is compiled into an executable code but in Java, a program gets compiled into an intermediate code called bytecode. This bytecode then gets executed by Java runtime system called the JVM. So, in order to run your Java program, you should have only JVM installed. JVM is also considered as the Interpreter of bytecode. In this way, Java has solved the problem of portability.

Robust

Generally, programs fail for the following two reasons:

- **Memory management:** Lack of memory management for allocating and deallocating memory for objects.

- **Exceptional Conditions:** Occurrence of unexpected exceptions.

Considering the preceding reasons, Java was designed to provide the ability to create robust programs. Therefore, Java compels the programmers to resolve the mistakes during compilation. Moreover, Java also frees the programmer from the worries of common causes of programming errors as the program code is verified during compilation.

Java also eradicates the problem of memory management by performing automatic garbage collection. It manages the memory allocation and deallocation for unused objects.

Multithreading

Java is a programming language that is designed for the distributed environment of the Internet and for that the concept of multithreading is implemented. This feature helps to write interactive programs wherein multiple tasks can be performed simultaneously; thus making it a robust programming language.

Java is inherently multi-threaded as there can be multiple executing threads in a single Java program. For example, a Java programmer can run three Java applets on the same page, provided that each applet gets equal time from the CPU.

The multi-threaded feature of Java makes it very responsive and robust. It is very important to note that, since Java is highly multi-threaded, thereby it is difficult to port it to architectures, such as Windows 3.1 or PowerMac, which do not natively support multi-threading.

Platform Independent

It is one of the most important features of Java as it is the first programming language, which is not bound to any specific operating system. In other words, Java programs can be executed anywhere on any system. No other programming language, before the evolution of Java, was platform independent and therefore, it can be said that Java is a revolutionary technology. Apart from cross-platform in source form, Java is also platform-independent in complied binary form.

A Java program does not execute natively on the host machine, instead Java interpreter reads the bytecode and executes the corresponding native machine instructions. Java interpreter is a special naïve program, which reads the bytecode, an intermediate form after compilation. Therefore, in order to port a Java program to a new platform, you need the interpreter and some of the library routines. Moreover, the Java compiler is also written in Java and the bytecodes are precisely defined, which remain same on all platforms.

Distributed

Java is a distributed language as it can be used to create applications to communicate over the network. Java can communicate over the network because it supports `TCP/IP` (Transmission Control Protocol/Internet Protocol). The TCP/IP protocol is a set of network communication protocol.

Dynamic

During runtime of a Java program, the relevant information is required that is used to verify and resolve access to objects. This concept of providing runtime information is referred to as dynamically linking the code. This feature of Java adds strength to the applet environment, in which small fragments of bytecode are dynamically updated on a running system.

Now after understanding general features of Java, let's study the new features of J2SE 5.0.

1.5 Exploring New Features of J2SE 5.0

The new Java SE 5 comes with lots of improvements to the existing packages with a number of new methods and interfaces added to it. The new version of API supports various new features which are being described under the following heads:

- Generics
- Enhanced for Loop
- Autoboxing and Unboxing
- Typesafe Enums
- Varargs

Generics

Java 5 introduces compile-time type safety to the Collection framework in the form of generics. This helps the programmers to not to provide the code for type casting. When you take an element out of a Collection, you must cast it to the type of the element that is stored in the collection. Moreover, it is unsafe to type cast an element as the compiler does not check the type cast during compilation. Therefore, Java 5 introduces generics, which provides a way to communicate the type of a collection to the compiler, so that the compiler can check it.

Enhanced for Loop

Initially for iterating over a collection by using for loop, you need to specify the three expression of for loop, as shown in the following code snippet:

```
void cancelAll(Collection<TimerTask> a) {
  for (Iterator<TimerTask> b = a.iterator(); b.hasNext(); )
  b.next().cancel();
}
```

However, now Java 5 introduces the enhanced for loop, as shown in the following code snippet:

```
void cancelAll(Collection<TimerTask> a) {
  for (TimerTask t : a)
  t.cancel();
}
```

The enhanced for loop has simplified the task of programmers as they need not to provide the three expressions for the for loop. This also results in simplification of the code.

Autoboxing and Unboxing

Initially the Java programmers need to provide the code for converting the primitive data types, such as int and the wrapper types, such as Integer. With the evolution of Java 5, a new feature of autoboxing and unboxing is introduced. The autoboxing and unboxing feature automates the process of typecasting the primitive data types and the wrapper types. This facilitates the Java programmers, as they do not have to provide code for type conversion.

■ Typesafe Enums

Initially you need to declare enums, as shown in the following code snippet:

```
public static final int FLOWER_ROSE = 0;
public static final int FLOWER_LILY = 1;
public static final int FLOWER_SUNFLOWER = 2;
public static final int FLOWER_JASMINE = 3;
```

The type of enum declaration shown in the preceding code snippet, resulted in problems, such as lack of type safety, lack of namespaces. However, with the evolution of Java 5, the type safe enums are declared, as shown in the following code snippet:

```
enum Flower{ ROSE, LILY, SUNFLOWER, JASMINE }
```

In the preceding code snippet, the `Flower` enum declared is type safe.

■ Varargs

Java 5 provides the varargs feature, which eliminates the need to manually box up argument lists into an array, while invoking methods that accept variable. Initially, you need to declare an array for a method that accepts an arbitrary number of values and then you need to put the values into the array, before invoking the method. The varargs feature automates and hides this entire process.

Now after understanding the new features introduced with the release of Java 5, let's discuss the Java environment.

1.6 Introducing the Java Environment

The Java environment comprises of numerous development tools, classes, and methods. The development tools for Java are provided as a part of the system known as Java Development Kit (JDK). Moreover, the Application Programming Interface (API) comprises of the classes and interfaces used in Java programming. Here, we discuss the following:

- Java Development Kit
- Java Platform Components
- Types of Java programs

Let's study each of them in detail.

■ Explaining JDK

The JDK is a Sun Microsystems product, which is aimed at Java developers. The JDK provides a collection of the Java tools, which are used while developing and running Java programs. The JDK comprises the following development tools:

- **java:** Serves as a Java interpreter used to run Java applets and applications by reading and interpreting bytecode files.
- **javac:** Serves as a Java compiler used to translate Java source code to bytecode files.
- **javacdoc:** Creates HTML documentation for Java source code files.

- **javap:** Serves as a Java disassemble used to convert bytecode files into a Java program description.
- **jdb:** Serves as a Java debugger used to find errors in Java programs.
- **appletviewer:** Facilitates to run Java applets.
- **jar:** Serves as archive used to package related class libraries into a single executable JAR file. This tool also helps to manage the JAR files.
- **javah:** Serves as the C header and stub generator, which is used to write native methods.

Explaining Java Platform Components

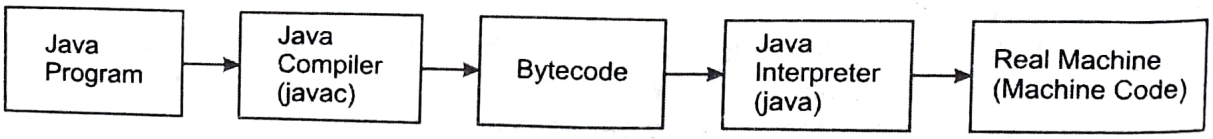
Sun Microsystems provide various Java editions to develop and run Java programs. The Java platform is not bound to any one processor or operating system, but the execution engine (virtual machine) and a compiler can run identically on any operating system. The following are the different editions of the Java platform:

- **The Java Platform, Standard Edition (Java SE):** The Java SE Edition of Java is the most commonly used among the three Java editions. It is used for developing desktop and console-based applications. It consists of the following:
  - A runtime environment
  - Set of Application Programming Interfaces (APIs) to build a wide variety of applications comprising standalone applications that can run on various platforms, such as Microsoft Windows, Linux, and Solaris
- **The Java Platform, Enterprise Edition (Java EE):** Java EE is a platform primarily used for building server-side applications, by using its convenient component-based approach.
- **The Java Platform, Micro Edition (Java ME):** Java ME enables you to build Java applications for micro-devices, which include handheld devices, such as mobile phones, PDAs, and other similar devices with limited display and memory support.

Here, we discuss the `Java SE`, which comprises of a runtime environment and API.

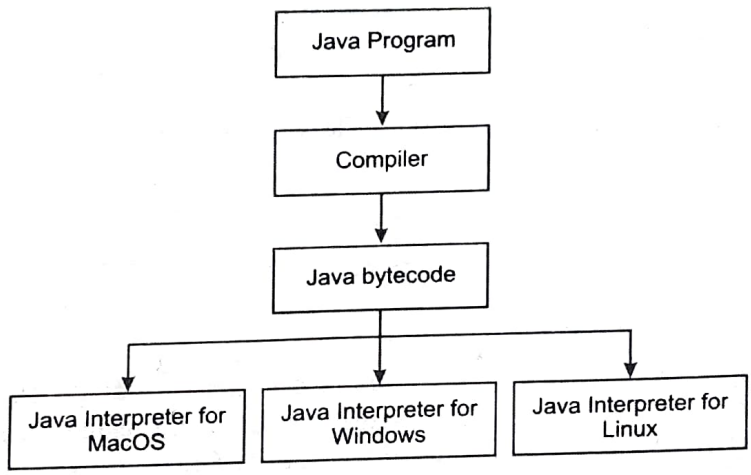
Defining JVM

In all programming languages, a compiler translates source code into machine code for a specific computer. In Java, a compiler translates the Java source code into an intermediate code known as bytecode for a virtual machine, known as Java Virtual Machine (JVM). Bytecode is the machine language for JVM and is not machine specific. Instead, the machine specific code is generated by the Java interpreter, which serves as a mediator between virtual machine and real machine. Figure 4 shows the process of compilation and conversion of bytecode into machine code:



▲ **Figure 4: Displaying Process of Compilation and Execution in Java**

In order to run a Java program on a different computer, you need a Java interpreter for Java bytecode. Although, Java bytecode interpreter is required depending upon the type of operating system in use, but once a computer has an interpreter, it can run a Java bytecode. In other words, the most important feature of Java is that the same compiled program can run on any operating system, as shown in Figure 5:



▲ **Figure 5: Displaying Process of Running Program on any Operating System**

Java itself is implemented as the Java Virtual Machine (JVM), which is the application that actually executes the Java program. When JVM is installed on a computer, it can run Java programs. Java programs; therefore, don't need to be self-sufficient, and they don't have to include all the machine-level code that actually runs on the computer. Instead, Java programs are *compiled* into compact bytecodes, and the JVM reads and interprets these bytecodes to run your program. To host Java on numerous computers, Sun only had to rewrite JVM to work on those computers. Since your program is stored in a bytecode file, it will run on any computer on which a JVM is installed.

Defining Java API

It is a collection of classes, interfaces, and methods which are provided in the form of Java packages. In other words, Java API is a large collection of already created classes and interfaces, which provide many useful capabilities, such as Graphical User Interface (GUI), Date, Time, and Calendar capabilities. The categorization of Java API in the form of Java packages is as follows:

- Application Programming Packages
  - **java.lang:** Provides classes that are fundamental to the design of the Java programming language.
  - **java.util:** Provides legacy collection classes, event model, collection framework, date and time capabilities, internationalization, and other utility classes, such as string tokenizer.
  - **java.io:** Provides classes for system input and output through data streams, serialization, and the file system.
- Applet and Network Programming Packages
  - **java.awt:** Provides classes for creating user interface and painting graphics and images.
  - **java.applet:** Provides classes that are necessary for creating an applet and those which are used to communicate with its applet context.
  - **java.net:** Provides classes that are used for implementing networking in Java programs.

Now after understanding the Java components, let's discuss the types of Java programs.

Types of Java Programs

A Java programmer can either create a Java application or a Java applet depending upon the requirement. In Java, you create the following types of programs:

- Applications
- Applets

Now let's discuss each of them in detail.

Defining Java Applications

A Java application is a program that is created by using Java programming language. Java applications are console-based, Character User Interface (CUI) based, and Graphical User Interface (GUI) based standalone applications. Once you become used to create applications, you will find that creating an application using Java is also as simple as creating application with any other programming language. In order to execute a Java application, you need to perform the following steps:

- Compile the Java source code using `javac` compiler to translate the source code into bytecode.
- Execute the bytecode by using the java interpreter.

Apart from applications, you can also create applets in Java.

Defining Java Applets

An applet is a program written in the Java programming language that can be included in an HTML page in the same way as an image is included in a page. An applet can be used in both `static` and `dynamic` Web pages to either display content or share information through the pages. You need to import the `java.applet` package that provides classes to either create an applet or allow an applet to communicate with its applet context.

The applet context is an application that is responsible for loading and running applets. For example, the applet context could be a Web browser or an applet development environment. In other words, an applet is a Java application designed to be transmitted over the Internet and is executed by a Java compatible Web browser. After understanding the types of Java programs, let's create a simple Java program. However, before creating a Java program, you must know that to create a Java program, you need to create a source code file using a text editor. After creating the Java program, you need to compile it using the Java compiler javac, and execute using the Java interpreter, java. The compiler generates a .class file by translating the source code into bytecode, which is then executed by the Java interpreter.

1.7 Developing a Simple Java Program

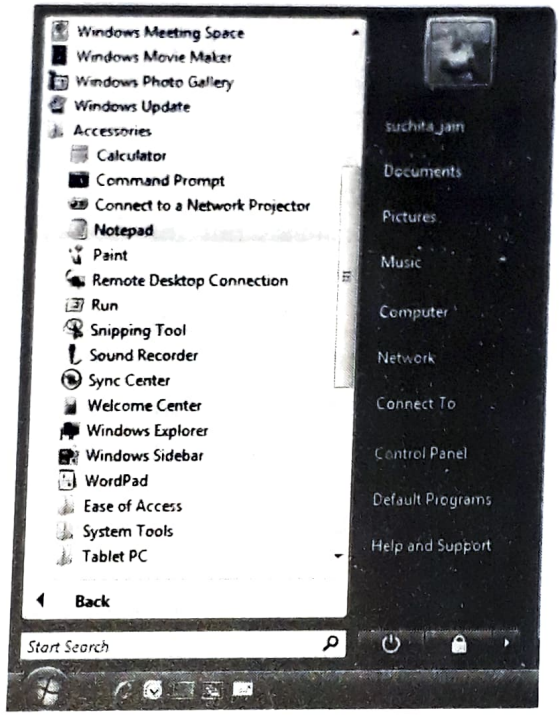
Here, let's create a simple program, which displays a hello message. Perform the following steps to develop a simple Java program:

- Writing the Java Program
- Saving the Java Program
- Compiling the Java Program
- Executing the Java Program

Writing the Java Program

You need to create a Java source file by using the plain text editor. The vi or pico editors are used to type programs in Unix system whereas in Windows, the notepad or DOS editor is used. Follow these steps to create a simple Java program:

1. Click Start→All Programs→Accessories→Notepad, as shown in Figure 6:



▲ **Figure 6: Opening a Notepad**

After clicking the Notepad, a blank notepad window is displayed.

2. Now, type the following code snippet in the blank Notepad window:

```
/* This is a simple Hello Java Program
which displays a Hello message */
```

The preceding code snippet is a comment in Java. The contents of comments are ignored by the Java compiler as comments only describe the operation of a Java program. Java permits both single line and multiple line comments. The single line comments begins with // and multiple line comments begins with /* and ends with */.

3. Now after providing comments, you need to declare the Java class, as shown in the following code snippet:

```
class Hello
{
}
```

The preceding code snippet declares the `Hello` class. The keyword class is used to declare a new class followed by the name of the class to be declared. As Java is an object-oriented programming language, therefore everything must be placed within the class. You must also note that in Java every class definition begins with an opening brace {and ends with the

matching closing brace}. This is similar to C++ class construct except a semicolon, which is provided towards the end in the C++ class declaration.

4. Now, we declare the `main()` method within the Hello class, as shown in the following code snippet:

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

In the preceding code snippet, the `main()` method is declared, which is similar to the `main()` function in C++. In Java, every program must include the `main()` method as it is the starting point for the Java interpreter to start the execution of a program. A Java application can have numerous classes but the `main()` method must be declared in only one of the classes to initiate the execution.

In the preceding code snippet, it is important to understand the public, static, and void keywords. The explanation of these keywords is as follows:

- **public:** It is an access specifier that allows Java programmers to control the visibility of class members. The public access specifier used before a class member implies that the class member may be accessed outside the class in which it is declared. Here, the `main()` method is declared as public implying that it can be called outside a class, when a program is started.
- **static:** It allows the `main()` method to be invoked without creating an instance of the class. It is essential to use the static keyword before the `main()` method as it is invoked by the Java interpreter before objects are created.
- **void:** It implies that the `main()` method does not return any value.

Moreover, you must have noticed that the `main()` method is declared with the parameters provided within the parenthesis. Here, `String args[]` declares a parameter named `args`, which contains an array of objects of the String class. The `args` parameter receives the command-line arguments, when the program is executed.

Now, provide the following code within the main() method, declared in the `Hello` class, as shown in the following code snippet:

```
System.out.println("Welcome to the world of programming");
```

The preceding code snippet is used to print the string "Welcome to the world of programming", followed by a new line on the screen. As Java is an object oriented language, thereby every method must be part of an object. Here, `println()` method is a member of the out object, which is a static data member of the System class. It is a predefined class that provides access to system and out is the output stream that is connected to the console.

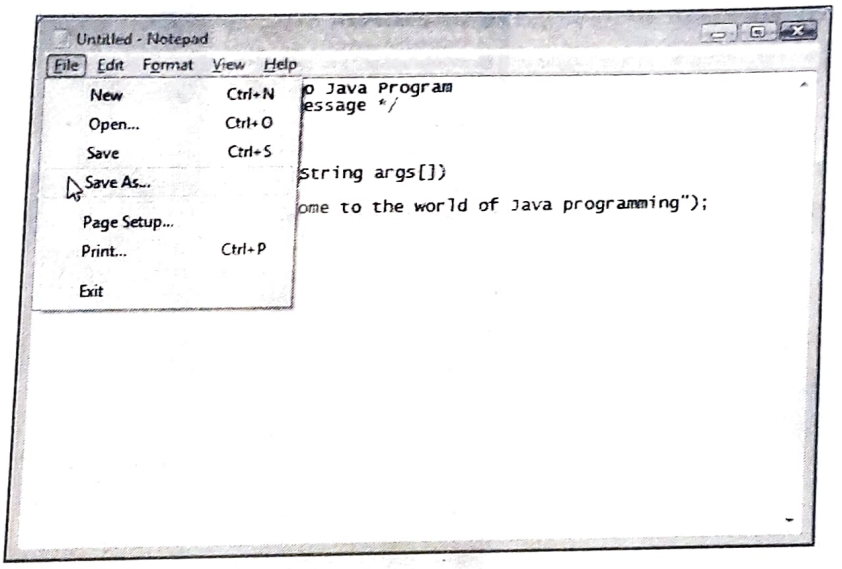
In Java, all statements end with a semicolon, for example the println() statement in the preceding code snippet.

Now, after writing the simple Java program, let's save the Java source code.

Saving the Java Program

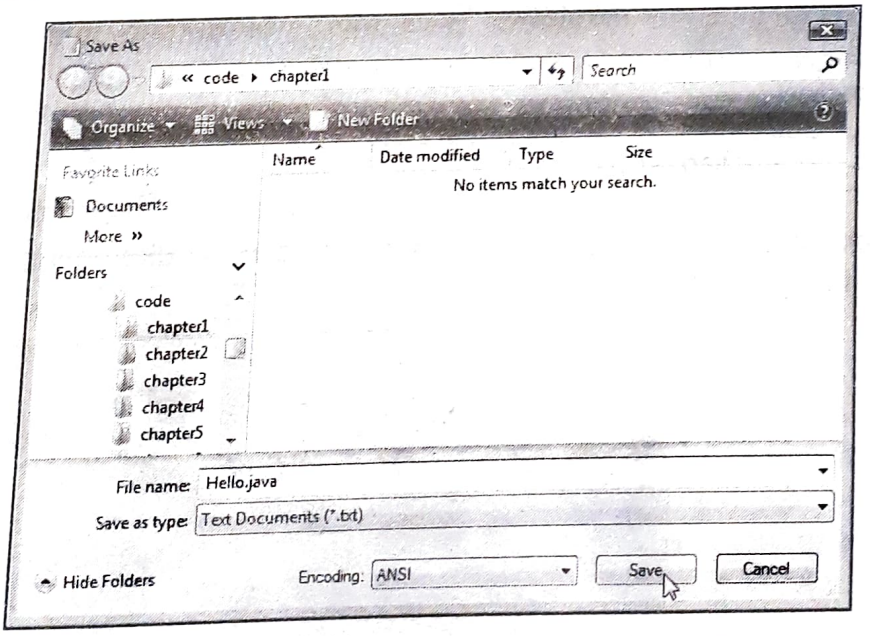
Before saving the Java program, you must be familiar with the fact that the source file name must match with the class name in which the main() method is invoked in the source code. In the preceding section, we have created the `Hello` class, so we need to save the source file with the name `Hello` followed by the java extension. Follow these steps to save the source file created in the preceding section:

1. Click Save As option from the File menu, as shown in Figure 7:



▲ **Figure 7: Saving the Java Source File**

After clicking the Save As option, the Save As Dialog box appears, as shown in Figure 8:



▲ **Figure 8: Displaying the Save As Dialog Box**

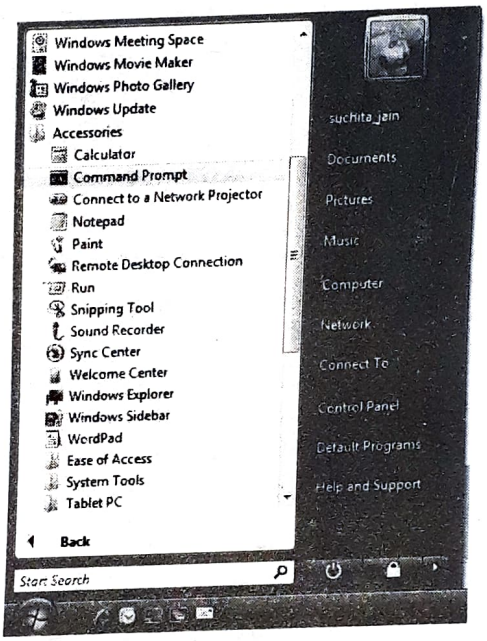
2. Select the code/chapter1 location in your computer to save the Java source file.
3. Write File name as Hello.java.
4. Then, click the Save button to save the Hello.java at the code/chapter1 location.

Let's now compile and execute this program to get the desired output.

Compiling the Java Program

In order to see the desired output of the simple Java program, you first need to compile the program. Firstly, the javac compiler translates the source code to bytecode, then java interpreter reads the bytecode and displays the desired output. Perform the following steps to compile the `Hello.java` source file:

1. Click Start→All Programs→Accessories→Command Prompt, as shown in Figure 9:



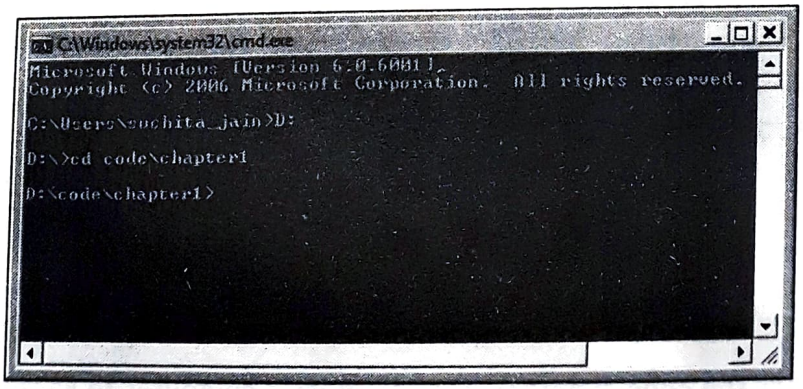
▲ **Figure 9: Opening the Command Prompt Window**

A new Command Prompt window appears.

2. Change the drive to D: and then type the following command to go to the code\chapter1 directory:

```
cd code\chapter1
```

After executing the preceding command, the current directory changes to code\chapter1, as shown in Figure 10:



▲ **Figure 10: Displaying the Current Directory as code\chapter1**

3. Execute the following command to compile the `Hello.java` file:

```
javac Hello.java
```

In the preceding command javac is compiler followed by the name of the Java source file. Hello.java. After executing the preceding command, the `Hello.class` file is created in the current directory code\chapter1.

Now, after compiling and generating the .class file of the Hello.java source file, let's execute the simple java program.

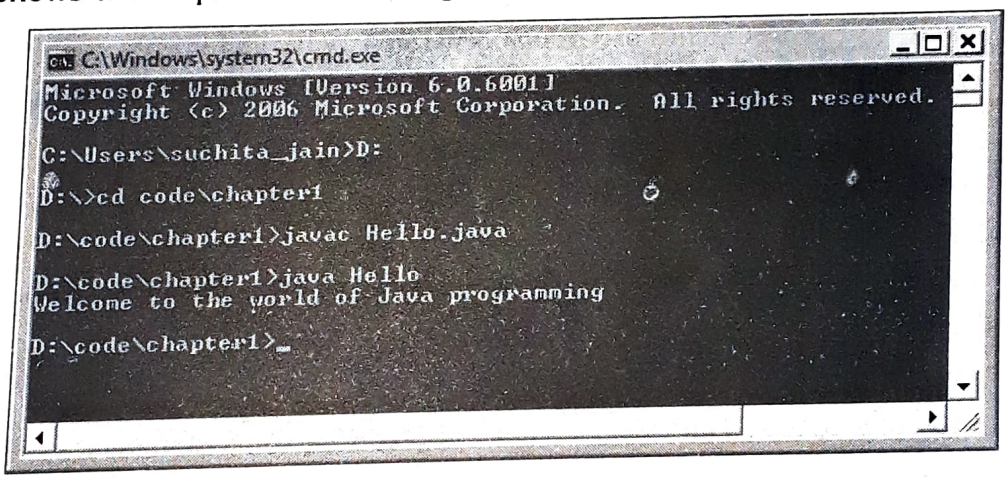
Executing the Java Program

The javac compiler has translated the source code of the Hello.java file into the `bytecode`. Now, the java interpreter helps in executing the Java program. Execute the following command to run the Java program:

```
java Hello
```

In the preceding command, java is interpreter followed by the name of the bytecode file, `Hello`.

Figure 11 shows the output after executing the preceding command:



▲ **Figure 11: Displaying the Output of Simple Java Program**

Now, after creating, compiling, and executing a simple Java program, let's discuss the Java tokens comprising of keywords, identifiers, literals, and operators.

1.8 Working with Java Tokens

The smallest individual element in a Java program is known as `token`. A Java program is a collection of classes which contain a set of declaration statements and methods containing executable statements. Most of the statements contain expressions, which describe the actions performed on data. The compiler recognizes tokens for building up expressions and statements. The Java programming language provides the following tokens:

- Keywords
- Identifiers
- Literals
- Constants
- Separators
- Operators

Keywords

Keywords are reserved words that cannot be used while naming a variable or method. Table 4 describes certain Java keywords:

Table 4: Java Keywords

Keywords	Description
abstract	Specifies that a class or method will be implemented later in a subclass.
assert	Evaluates the expressions without writing the code for exception handling.
boolean	A data type that holds true and false values only.
break	A control statement for breaking out of loops.
byte	A data type that holds 8-bit data values.
byvalue	It is reserved for future use.
case	Used in switch statements to mark blocks of text.
cast	It is reserved for future use.
catch	Catches exceptions generated by try statements.
char	A data type that holds unsigned 16-bit Unicode characters.
class	Declares a new class.
const	It is reserved for future use.
continue	Sends control back to the top of a loop.
default	Specifies the default block of code in a switch statement.
do	Starts a do-while loop.
double	A data type that holds 64-bit floating-point numbers.
else	Indicates the alternative branch in an if statement.
extends	Indicates that a class is derived from another class or an interface is derived from another interface.
enum	A class of constants that represents fixed choices.
final	Indicates that a variable holds a constant value or that a method will not be overridden.
finally	Indicates a block of code in a try-catch structure that is always executed.
float	A data type that holds a 32-bit floating-point number.
for	Used to start a 'for' loop.
goto	It is reserved for future use.
if	Tests a true/false expression and branches execution accordingly.
implements	Specifies that a class implements an interface.
import	References other classes.
instanceof	Indicates whether an object is an instance of a specific class or implements a specific interface.
int	A data type that holds a 32-bit signed integer.
interface	Declares an interface.

Table 4: Java Keywords

Keywords	Description
long	A data type that holds a 64-bit integer.
native	Specifies that a method is implemented with the native (platform-specific) code.
new	Creates new objects of a class.
package	Declares a Java package.
private	An access specifier that indicates a method or variable that can be accessed only in the class where it has been declared.
protected	An access specifier indicating that a method or variable can be accessed either in the class where it has been declared or in a subclass of the class where it has been declared or in other classes of the same package.
public	An access specifier used for classes, interfaces, methods, and variables indicating that an item is either accessible throughout the application or accessible where the class is defined.
return	Sends control and returns a value back from a called method.
short	A data type that holds a 16-bit integer.
static	Indicates that a variable or method is a class method instead of being restricted to a particular object.
strictfp	Retrieves the consistent floating point values that are independent of platforms.
super	Refers to the base class of a class (used in a method or a class constructor).
switch	A statement that executes code based on a test value.
synchronized	Specifies critical sections or methods in a multithreaded code.
this	Refers to the current object in a method or constructor.
throw	Creates an exception.
throws	Indicates what possible exceptions can be thrown by a method.
transient	Specifies that a variable is not part of an object's persistent state.
try	Starts a block of code that will be tested for exceptions.
var	It is reserved for future use.
void	Specifies that a method does not have a return value.
volatile	Indicates that a variable can change asynchronously.
while	Starts a while loop.

The keywords discussed in Table 4 should not be used as identifiers in Java. The following section explains what are identifiers and rules for legal identifiers.

Identifiers

An identifier is a symbolic name that refers to the name of variables, methods, classes, packages, or interfaces in a Java program. Each variable is identified by its name. The variable name should relate to its purpose. Similarly, in case of a method, the name of the method

should relate to the method's functionality. Java identifiers consist of alphabets, numbers, underscore (_), and the dollar sign ($). Now, let's discuss about the legal identifiers in a Java program.

Legal Identifiers

Legal identifiers are identifiers that conform to the rules that Java compiler uses to determine whether or not the name is legal. Some rules for legal identifiers are as follows:

- Must start with an underscore (_), a letter, or a dollar sign ($)
- May consist of a letter, a number, an underscore, and a dollar sign
- Does not allow spaces in between an identifier
- Cannot have a name that is similar to a Java reserved keyword
- Are case-sensitive, that is, the word salary, Salary, and SALARY are three different identifiers

Using Literals

In a Java program, you can assign a literal value to a variable. Literal refers to the constant values that can be changed depending on the program requirement. The source code representations of the values of primitive data types in a Java program are known as primitive literals. These primitive literals can be an integer number, floating-point number, a `boolean` value, or a character that you use in your program. Some examples of the primitive literals are:

```
24          //an int literal
124.543     // a double literal
true        // a boolean literal
'v'         // a char literal
```

In the preceding example, 24 is an integer value, 124.543 is a floating point number, true is the boolean literal, and v is a character value. Now, let's discuss the following literals in detail:

- Integer Literals
- Floating-Point Literals
- Boolean Literals
- Character Literals
- String Literals

Integer Literals

Based on the requirement of the program, an integer number can be represented in the following three ways:

- `Decimal` (base 10)
- `Octal` (base 8)
- `Hexadecimal` (base 16)

Decimal Literals

The decimal literals are commonly used to represent numbers in real life. They have base 10 and use digits between 0-9 to represent all numbers. The decimal representation of an integer number does not use any prefix as compared to other forms of representing integers. Listing

1 shows representation of the decimal literal (you can find the DecimalRep.java file on the CD in the code\chapter1 folder):

▶ **Listing 1: Decimal Representation**

```
class DecimalRep {
  public static void main(String args[]) {
      int sixteen=16;
      System.out.println("Decimal Sixteen="+sixteen);
  }
}
```

In Listing 1, an integer value (16) is represented in decimal notation, which is assigned to an `int` type of variable `sixteen`. Here, you can notice that no prefix is used with (16); however, in the case of octal and hexadecimal literals, the prefix is used.

Octal Literals

The other way of representing integer numbers is octal representation, which has base 8. It uses only 8 digits, from 0 to 7, to represent all the numbers. In Java, the octal numbers are preceded with a prefix 0, which notifies the compiler that the number is represented in the octal form. Listing 2 shows representation of the octal literals (you can find the OctalRep.java file on the CD in the code\chapter1 folder):

▶ **Listing 2: Octal Representation**

```
class OctalRep {
  public static void main(String args[]) {
  int five=05;          // value of five is equal to decimal 5
  int eight=010;       // value of eight is equal to decimal 8
  int ten=012;        // value of ten is equal to decimal 10
  System.out.println("Octal value 012 is equivalent to "+ten+" in decimal");
  }
}
```

In Listing 2, `five`, `eight`, and `ten` are integer variables, which are being assigned with `octal` literals. Notice a prefix 0 (zero) in all the octal numbers, which intimates the JVM that the numbers assigned are in octal representation.

TEST YOUR KNOWLEDGE

Q1. Write a program to demonstrate octal representation of an integer number.

Ans.

```
/* program to demonstrate octal representation of an integer number */
class TestYourKnowledge1
{
  public static void main(String args[])
  {
  int five=05;          // value of five is equal to decimal 5
  int eight=010;       // value of eight is equal to decimal 8
  int ten=012;         // value of ten is equal to decimal 10
  System.out.println("Octal value 012 is equivalent to "+ten+" in
      decimal");
  }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge1.java
D:\code\chapter1>java TestYourKnowledge1
```

Output:

```
Octal value 012 is equivalent to 10 in decimal
```

Hexadecimal Literals

The hexadecimal representation of an integer number has base 16. It uses 10 digits and 6 special symbols/characters to represent numbers. The digits used by hexadecimal notation are 0 to 9 and the special symbols used are a, b, c, d, e, and f. The representation of 0-9 in the hexadecimal notation is the same as in the decimal notation and the remaining 6 are represented by characters a to f. The respective representations of 10 to 15 in the hexadecimal notation are:

- 10 – a or A
- 11 – b or B
- 12 – c or C
- 13 – d or D
- 14 – e or E
- 15 – f or F

An important aspect that must be taken into consideration while representing integers in the hexadecimal notation is the prefix 0x (zero with x). All hexadecimal representations must start with 0x or 0X to intimate the JVM that the given number is in hexadecimal form. Listing 3 shows representation of the integer numbers in the hexadecimal notation (you can also find the HexaDecimalRep.java file on the CD in the code\chapter1 folder):

▶ **Listing 3: Hexadecimal Representation**

```
class HexaDecimalRep {
public static void main(String ar[]) {
  int a=0x5;    // value of a is equivalent to decimal 5
  int b=0XF;    // value of b is equivalent to decimal 15
  int c= 0xd;   // value of c is equivalent to decimal 13
  System.out.println("Decimal five= "+a);
  System.out.println("Decimal fifteen= "+b);
  System.out.println("Decimal thirteen="+c);
  }
}
```

By default, all the three integer types (decimal, octal, and hexadecimal) are considered to be of `int` type, but we can also assign these integer literals to long type of variables with a trailing l or L, as shown in the following code snippet:

```
long distance=65347L;    // a decimal integer literal is assigned
long length=0xCCCCl;     //a hexadecimal integer literal is assigned
```

TEST YOUR KNOWLEDGE

Q2. Write a program to demonstrate hexadecimal representation of an integer number.

Ans.

```
/* Program to demonstrate hexadecimal representation of an integer
 number */
class TestYourKnowledge2
{
  public static void main(String ar[]) {
      int a=0x5; // value of a is equivalent to decimal 5
      int b=0XF; // value of b is equivalent to decimal 15
      int c= 0xd;// value of c is equivalent to decimal 13
      System.out.println("Decimal five= "+a);
      System.out.println("Decimal fifteen= "+b);
      System.out.println("Decimal thirteen="+c);
  }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge2.java
D:\code\chapter1>java TestYourKnowledge2
```

Output:

```
Decimal five= 5
Decimal fifteen= 15
Decimal thirteen=13
```

Now, after discussing all the three forms of representing Integer literals, let's discuss the floating-point literals, which provide fractional values to variables.

➡ Floating-Point Literals

In a Java program, the whole numbers are represented as integers but a fractional number is represented and assigned using `floating-point` values. The source code representation of floating-point values in a Java program is called floating-point literals. A floating point number consists of integer numbers and a decimal point. Consider the following example:

```
double distance=43213.13245;
```

Here, we have represented a floating-point literal as 43213.13245 in which the first part is 43213, followed by a decimal point (.), and the second part is 13245.

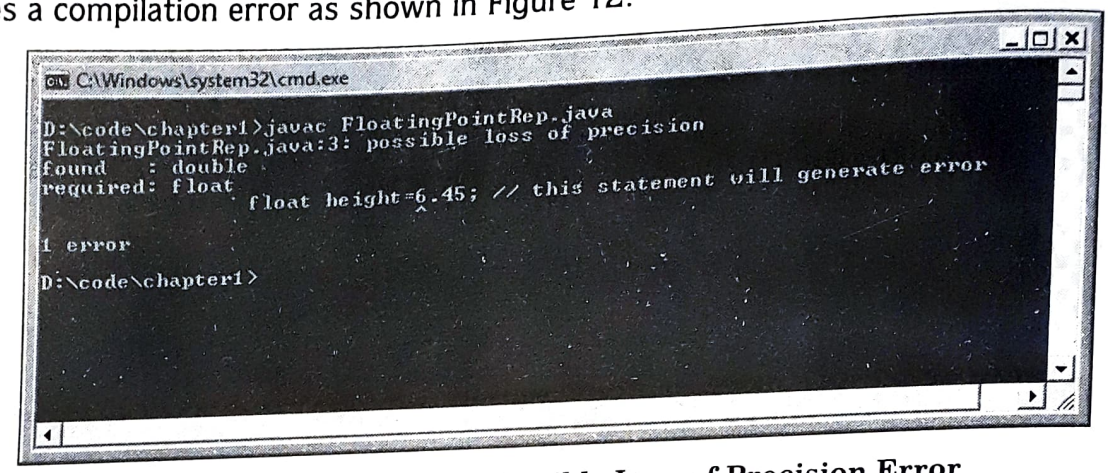
In Java programming language, by default, the floating-point literals are defined as double that has a width of 64 bits. You can assign a floating-point literal in a float type of variable but it will generate compilation error. The error states the possible loss of precision as the width of float data type is 32 bits.

Listing 4 shows an example where a compilation error occurs when a float type of variable is assigned with floating-point literals without adding F postfix (you can find the FloatingPointRep.java file on the CD in the code\chapter1 folder):

Listing 4: Floating-Point Representation through Float Data Type

```
class FloatingPointRep {
  public static void main(String ar[]) {
      float height=6.45; // this statement will generate error
      System.out.println("Height= "+height);
  }
}
```

Save Listing 4 as the FloatingPointRep.java source file and during compilation, the source file generates a compilation error as shown in Figure 12:



▲ **Figure 12: Displaying Possible Loss of Precision Error**

Figure 12 shows that the compiler generates an error showing that the literal assigned to the height variable needs to be float instead of double that has been assigned to it. The compilation error stating the possible loss of precision occurs because the variable on left has narrow width than the literal being assigned in it.

To avoid such compilation errors, you can explicitly take the responsibility of loss of precision in your program, which means that you can add F or f at the end of floating-point literal assigned. When the compiler comes across the floating-point literal with a postfix F or f, it concludes that the programmer is aware of the possible loss of precision and therefore the compiler does not show any error. Listing 5 shows the correct implementation of the floating-point literals through float data type:

Listing 5: Floating-Point Representation through Float Data Type

```
class FloatingPointRep {
  public static void main(String ar[]) {
      float height=6.45f;
      System.out.println("Height= "+height);
  }
}
```

When you change the code of the `FloatingPointRep.java` file with the code provided in Listing 5, then no compilation error occurs. In Listing 5, f is added at the end of the floating-point literal 6.45 to explicitly intimate the compiler that the programmer is aware of the possible loss of precision, so that it does not generate any compilation error.

Similarly, you can also specify D or d to a floating-point literal while assigning it to a `double` data type, but it is optional because by default the floating-point literals are double values. The

following code snippet shows an example, in which D is added while assigning a double value to the height variable:

```
double height=67582.4545D;
double height=67582.4545;
```

In the preceding code snippet, both the representations are same and correct, although D has not been used in the second representation. If a floating-point literal contains a comma (,) in place of decimal point (.) then it is incorrect. Consider the following code snippet:

```
double distance=1245,675;   //this representation is incorrect and the
  compiler will generate an error
```

Apart from integer and floating-point literals, we also have `boolean` literals which can have either true or false as their value. Now, let's discuss the `boolean` literals in detail.

Boolean Literals

In Java, `boolean` values are the result of conditions, which return either true or false. The source code representation of the `boolean` values is called boolean literals. The possible two `boolean` literals are true and false. You must know that unlike C and C++ programming languages, in Java, the `boolean` literals cannot be represented in the form of numbers as 0 for false and 1 or any other number for true. In other words, the Java programming language does not support any number for true and false `boolean` literals. Consider the following code snippet:

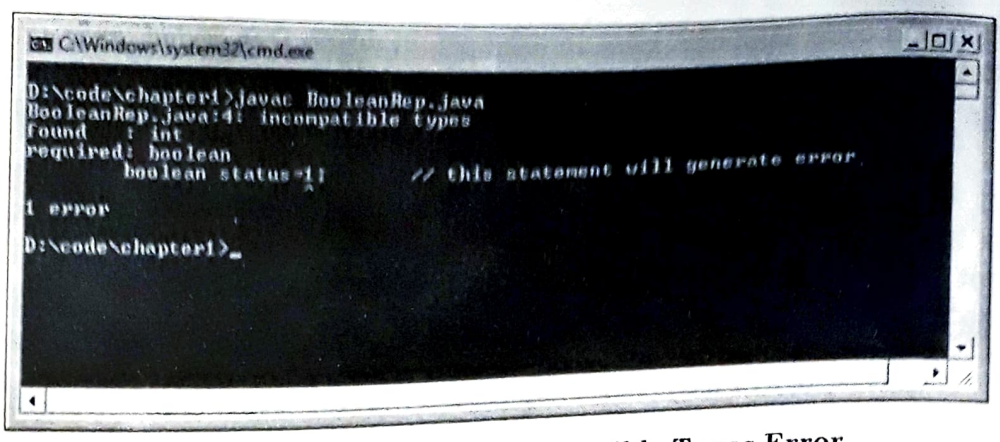
```
boolean status=true;     // no error
boolean status=1;    // will generate compilation error
```

In the preceding code snippet, the first representation of the `boolean` literal is correct and executes without any compilation error. However, the second representation of `boolean` literal is incorrect as 1 is assigned to a boolean data type leading to a compilation error. Listing 6 shows an example where a compilation error is generated due to incorrect representation of `boolean` literal (you can find the BooleanRep.java file on the CD in the code\chapter1 folder):

▶ **Listing 6: Boolean Literal Representation**

```
class BooleanRep {
  public static void main(String ar[]) {
  boolean status=1;  // this statement will generate error
  if(status) {
      System.out.println("Here Status= "+status);
  }
  }
}
```

Save Listing 6 as BooleanRep.java file and during compilation, it generates a compilation error as shown in Figure 13:



▲ **Figure 13: Displaying Incompatible Types Error**

Figure 13 shows the compilation error stating the incompatible types, that is, instead of assigning a `boolean` value to a `boolean` type variable, an `int` value is assigned.

To represent characters in a program, Java provides us with character literals. Let's now study about character literals in detail.

➡ Character Literals

In Java, the source code representation of a single character value is called character literal. The character literals in Java are represented by a single character or an escape sequence character enclosed within single quotes. Consider the following code snippet:

```
char letter='A';
char value='y';
char newline='\n';
```

Here, the character literals A, y, and \n (escape sequence character for a new line) are represented. It is essential to understand that the character literals can also be represented using their Unicode values. These Unicode values are prefixed with \u. Consider the following code snippet:

```
char character J='\u004A';
```

Here, you can see that character J is represented as a Unicode character. You must know that the characters are 16-bit unsigned integers and you must assign only those number literals for characters that fit into the unsigned 16-bit range, that is, 65535. Consider the following code snippet:

```
char characterA=65;// an integer literal representation of character A
char characterB=0102// an octal literal representation of character B
char characterC=0x43;   // a hexadecimal representation of character C
```

In the preceding code snippet, different values of integer literals are assigned to character literals and all are legal as they are written in unsigned 16-bit range. However, if you assign an integer literal which is out of unsigned 16-bit range, then the compiler generates error, provided that you have not explicitly typecast its value. Consider the following code snippet:

```
char illegalchar=67875;
```

During compilation, the preceding code snippet generates an error. If you assign an integer literal for a character, which is not in the unsigned 16-bit range, then the compiler must be notified explicitly by typecasting. For example, the following code snippet explicitly typecasts the integer literal to char:

```
char legalchar=(char)67875;
```

Here, the integer literal 67875 is out of the unsigned 16-bit range, so it is typecast by adding `char` before it. This typecast intimates the compiler that the programmer is aware of the possible loss of precision due to which compiler does not generate any error.

Apart from all the primitive literals discussed so far in this chapter, sometimes you may also come across String literals. Although String is not a primitive data type in Java, String values can be represented as literals. Let's now discuss String literals in detail.

Strings Literals

Unlike C++, in Java programming language, String is considered as a class instead of a primitive data type and thereby all the string literals are the values of the String class objects. The source code representation of a value of a String object is called String literal. The String literal is a collection of characters enclosed within double quotes. The following code snippet represents the String literals:

```
String msg="Welcome to Java Programming";
System.out.println(msg);
```

In the preceding code snippet, first, a `String` literal is assigned as a value to the String object `msg` and then the literal assigned is displayed on the standard output console.

TEST YOUR KNOWLEDGE

Q3. Write a program to display a welcome message.

Ans.

```
/*Program to display a welcome message*/
class TestYourKnowledge3
{
 public static void main(String args[])
 {
     System.out.println("Welcome to the world of Java Programming");
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge3.java
D:\code\chapter1>java TestYourKnowledge3
```

Output:

```
Welcome to the world of Java Programming
```

Explaining Constants

Sometimes the values assigned to variables changes throughout the execution of a program, however, it may be possible that you want to lock the assigned value to a variable. In other words, if you want a value to remain fixed throughout the execution of a program, then it is known as `constant`.

Let's consider a real life scenario, consider you want to calculate the area of a circle in your program. While calculating the area of a circle, the value of pi remains same irrespective of the change in radius. Therefore, you need to assign a fixed value, 3.14, as shown in the following code snippet:

```
final double pi = 3.14;
```

Here, in the preceding code snippet, the `pi` variable is assigned a fixed value 3.14. You must note that the final keyword is used to avoid the modifications in the value of pi variable. The final keyword in Java is similar to const in C/C++. You must remember the following points in context of the usage of final keyword:

- You should choose all uppercase identifiers for final variables, as it is a common coding convention.
- Once you have declared a final variable, you cannot assign any other value to that variable.
- In C and C++, the constants are defined using the # define statement; however, in Java, the constants are declared on the basis of their types.
- You cannot declare a final variable within a method, since they should be declared as class members towards the beginning of a class.
- The final variables are essentially a constant and they do not occupy memory on a per instance basis.

TEST YOUR KNOWLEDGE

Q4. Write a program to calculate the area of circle.

Ans.

```
/* Program calculating area of the circle */
class TestYourKnowledge4
{
public static void main(String args[])
{
final double pi= 3.14;
int radius = 10;
System.out.println("Area is:" + radius*radius*pi);
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge4.java
D:\code\chapter1>java TestYourKnowledge4
```

Output:

```
Area is:314.0
```

Q5. Write a program to calculate the volume of a hemisphere.

Ans.

```
/* Program to calculate the volume of a hemisphere */
class TestYourKnowledge5
{
 public static void main(String args[])
 {
     float radius =2;
     float volume;
     float pi = 3.14f;
     volume = (2 * pi * radius * radius)/3;
     System.out.println("Volume of Hemisphere is:" + volume);
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge5.java
D:\code\chapter1>java TestYourKnowledge5
```

Output:

```
Volume of Hemisphere is:8.373334
```

Explaining Separators

Separators are the symbols that are used to structure a Java program in a well defined manner. The usage of separators facilitate in grouping the code and arranging them in an appropriate manner. Table 5 describes the purpose of separators along with their symbols:

Table 5: Symbols and Purpose of Separators in Java

Name of the Separator	Symbol	Usage
Parentheses	()	Contains list of parameters while declaring or invoking methods. It is also used in adjusting precedence in arithmetic expressions, surrounding cast types, and containing expressions in control statements
Braces	{}	Contains values of automatically initialized arrays and is also used to define a block of code for classes, methods, and local scopes
Brackets	[]	Declares array types and also used in dereferencing array values
Semicolon	;	Terminates the statements provided in a Java program
Comma	,	Separates successive identifiers in variable declarations and is also used in chaining statements together inside a for loop

Table 5: Symbols and Purpose of Separators in Java

Name of the Separator	Symbol	Usage
Period	.	Separates a variable or a method from an object. For example, Car c: //creating instance of the Car variable c.go(); //separates the c instance from the go() method
Colon	:	Used after loop labels

Using Operators in Java

An operator is a symbol that allows a programmer to perform certain arithmetic or logical operations on data and variables. Here, we discuss the following operators in Java:

- Arithmetic
- Relational
- Boolean logical
- Increment and decrement
- Conditional
- Bitwise
- Special (instanceOf and dot)

Now, let's discuss all these operators and their implementation in detail.

The Arithmetic Operators

The operators that are used to perform arithmetic operations are called arithmetic operators such as addition (+), subtraction (-), and multiplication (*).Table 6 lists the arithmetic operators supported in Java:

Table 6: Arithmetic operators in Java

Arithmetic operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

The arithmetic operators are used to perform mathematical operations as in algebra. You can perform the addition and subtraction operations on either integers or double numbers. Listing 7 shows the addition and subtraction of integers (you can find the IntAddSub.java file on the CD in the code\chapter1 folder):

▶ Listing 7: Addition and Subtraction on Integers

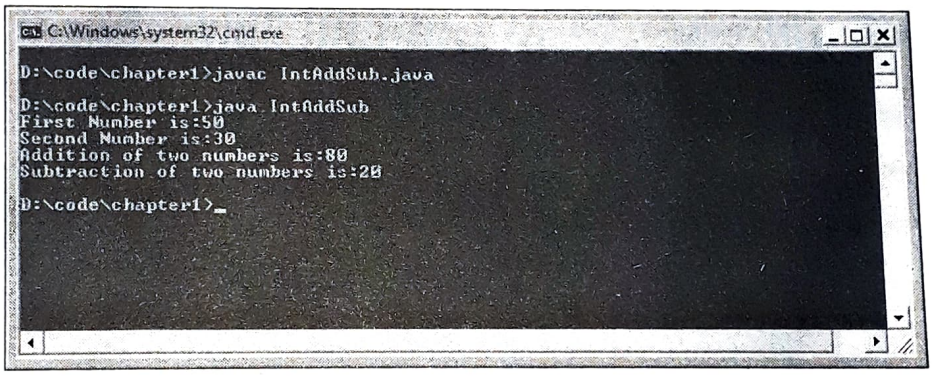
```
/* Adding and Subtracting Integer numbers in this program
*/ class IntAddSub {
 public static void main(String args[]) {
```

```
        int a = 50;
        int b = 30;
        System.out.println("First Number is:" +a);
        System.out.println("Second Number is:" +b);
        int c = a + b;
        System.out.println("Addition of two numbers is:" +c);
        c = a-b;
        System.out.println("Subtraction of two numbers is:" +c);
    }
}
```

In Listing 7, the two integer numbers a and b are added and the resultant value is displayed. Similarly, these two numbers are subtracted and their output is displayed. Save the Listing 7 at code\chapter1 as the IntAddSub.java source file. Now compile and execute the IntAddSub.java file by using the following commands:

```
javac IntAddSub.java
java IntAddSub
```

This first command compiles the source file and creates the IntAddSub.class file. In the second command, the java interpreter displays the output on the console, as shown in Figure 14:



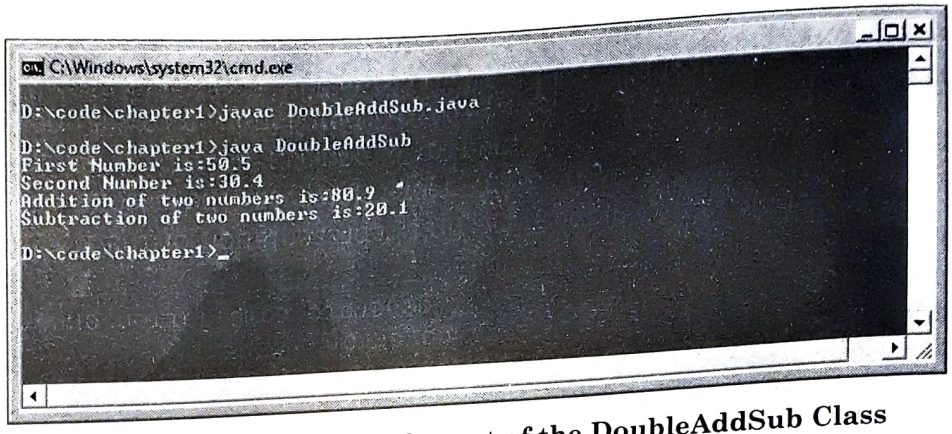
▲ **Figure 14: Displaying Output of the IntAddSub Class**

Now after understanding the addition and subtraction of integer numbers, let's now discuss the addition and subtraction of the double values. Listing 8 shows the addition and subtraction of the p and q double type variables (you can find the DoubleAddSub.java file on the CD in the code\chapter1 folder):

▶ **Listing 8: Addition and Subtraction of double Numbers**

```
/* Adding and Subtracting Double numbers in this program */
class DoubleAddSub {
  public static void main(String args[]) {
      double a = 50.5;
      double b = 30.4;
      System.out.println("First Number is:" +a);
      System.out.println("Second Number is:" +b);
      double c = a + b;
      System.out.println("Addition of two numbers is:" +c);
      c = a - b;
      System.out.println("Subtraction of two numbers is:" +c);
  }
}
```

In Listing 8, the a and b are double type variables, which are added and subtracted. The resultant value of their addition and subtraction is displayed as the output of Listing 8. Now, compile and execute the DoubleAddSub class and the output is displayed, as shown in Figure 15:



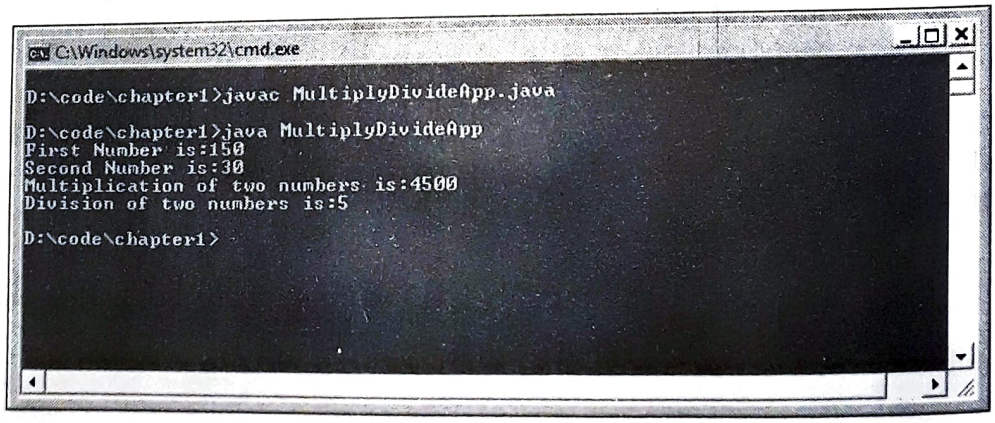
▲ **Figure 15: Displaying Output of the DoubleAddSub Class**

After understanding the addition and subtraction, let's discuss how the multiplication and division operations are performed in a Java program. Listing 9 shows the multiplication and division of two integer values (you can find the MultiplyDivideApp.java file on the CD in the code\chapter1 folder):

▶ **Listing 9: Multiplication and Division of Two Numbers**

```
/* Multiplying and Dividing two numbers in this program */
class MultiplyDivideApp {
 public static void main(String args[]) {
     int a = 150;
     int b = 30;
     System.out.println("First Number is:" + a);
     System.out.println("Second Number is:" + b);
     int c = a * b;
     System.out.println("Multiplication of two numbers is:" + c);
     c = a/b;
     System.out.println("Division of two numbers is:" + c);
 }
}
```

In Listing 9, two numbers are multiplied and divided, whose resultant value is displayed on the console. Figure 16 shows the output of the MutiplyDivideApp class:



▲ **Figure 16: Displaying Multiplication and Division of Two Numbers**

TEST YOUR KNOWLEDGE

Q6. **Write a program to calculate the average of three numbers.**

Ans.

```
/* Program calculating average of three numbers */
class TestYourKnowledge6
{

public static void main(String args[])
{
int x =10;
int y =20;
int z =30;
System.out.println("Average is:" + ((x + y+ z)/3));
}

}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge6.java
D:\code\chapter1>java TestYourKnowledge6
```

Output:

```
Average is:20
```

Q7. **Write a program to calculate square and cube of a number.**

Ans.

```
/* Program calculating square and cube of a number */
class TestYourKnowledge7
{

public static void main(String args[])
{
int x =3;
System.out.println("Square of a number is:" + x*x);
System.out.println("Cube of a number is:" + x*x*x);
}

}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge7.java
D:\code\chapter1>java TestYourKnowledge7
```

Output:

```
Square of a number is:9
Cube of a number is:27
```

Q8. Write a program to calculate the hypotenuse of right angled triangle when other sides of the triangle are given.

Ans.

```
/*Program to calculate the hypotenuse of right angled triangle given
 other
      sides of the triangle*/
class TestYourKnowledge8
{
public static void main(String args[])
{
double x = 3.0;
double y = 4.0;

double z = Math.sqrt(x*x + y*y);
System.out.println("Hypotenuse is:" + z);
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge8.java
D:\code\chapter1>java TestYourKnowledge8
```

Output:

```
Hypotenuse is:5.0
```

Q9. Write a program to evaluate simple interest of a given principle, rate, and time.

Ans.

```
/*Program to evaluate simple interest of a given principle, rate, and
     time*/
class TestYourKnowledge9
{
 public static void main(String args[]) {
     int p =1000;
     int r =5;
     int t =2;
     System.out.println("Simple Interest is:" + (p*r*t)/100);
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge9.java
D:\code\chapter1>java TestYourKnowledge9
```

Output:

```
Simple Interest is:100
```

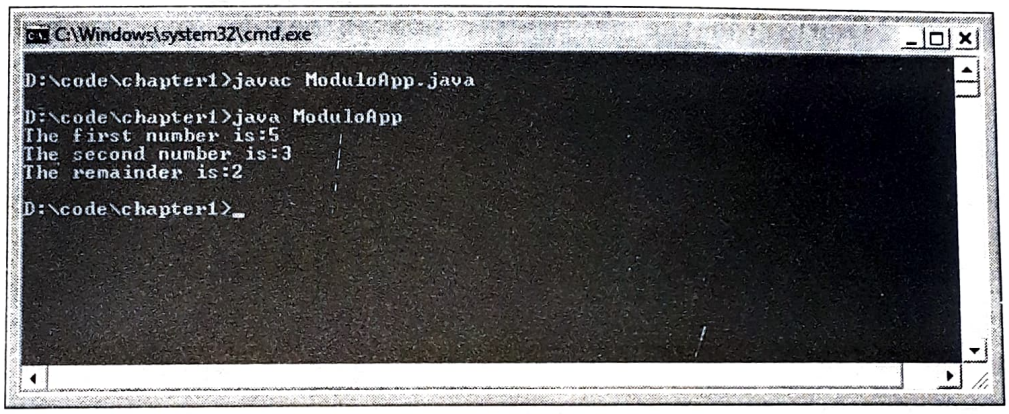
Now, the last arithmetic operator to be discussed is the `modulo` operator. Listing 10 shows the implementation of the `modulo` operator (you can find the ModuloApp.java file on the CD in the code\chapter1 folder):

▶ **Listing 10: The ModuloApp.java File**

```
/* Calculating the remainder left after dividing the two numbers */
class ModuloApp {
 public static void main(String args[]) {
     int p = 5;
     int q = 3;
     System.out.println("The first number is:" + p);
     System.out.println("The second number is:" + q);
     int r = p % q;
     System.out.println("The remainder is:" + r);
 }
}
```

NCS859

In Listing 10, the `ModuloApp` class declares two variables and displays their remainder after division. Figure 17 shows output of the `ModuloApp` class, after compilation and execution:



▲ **Figure 17: Displaying Remainder of Two Numbers**

TEST YOUR KNOWLEDGE

Q10. Write a program using the arithmetic operators to perform algebraic operations on two numbers.

Ans.

```
/*Program using the arithmetic operators to perform algebraic
  operations on two numbers*/
class TestYourKnowledge10
{
 public static void main(String args[])
 {
     int x = 20;
     int y = 10;
     System.out.println("Addition is:" + (x+y));
     System.out.println("Subtraction is:" + (x-y));
     System.out.println("Multiplication is:" + (x*y));
     System.out.println("Division is:" + (x/y));
     System.out.println("Remainder is:" + (x%y));
 }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge10.java
D:\code\chapter1>java TestYourKnowledge10
```

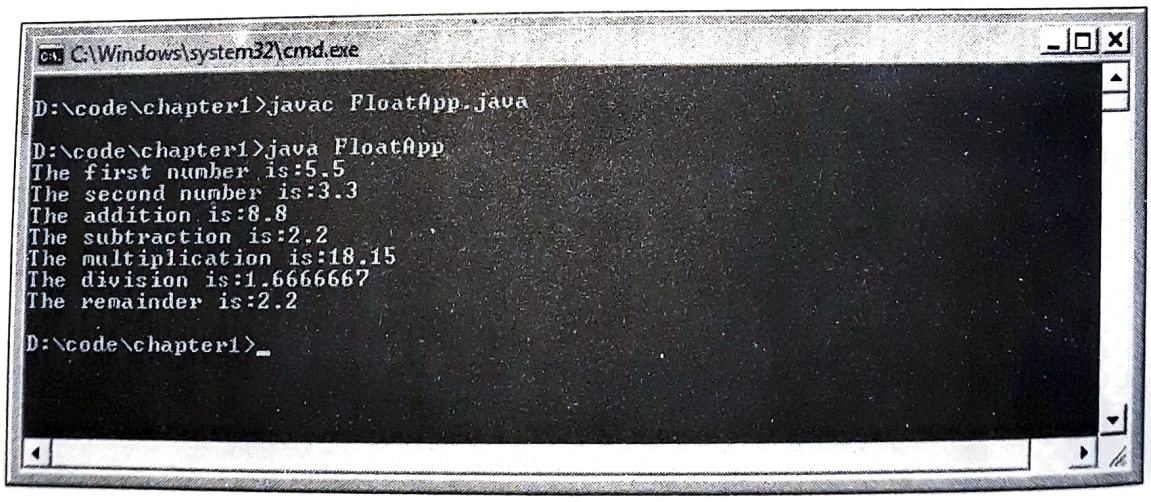
```
Output:

Addition is:30
Subtraction is:10
Multiplication is:200
Division is:2
Remainder is:0
```

Till now, we have created various programs to perform mathematical operations on integer and double numbers. Now, it's essential to understand the implementation of addition, subtraction, multiplication, division, and modulus on the float type variables. Listing 11 shows the floating point arithmetic operations (you can find the FloatApp.java file on the CD in the code\chapter1 folder):

▶ **Listing 11: The FloatApp.java File**

```
/* Calculating the arithmetic operations on the two float type numbers */
class FloatApp {
  public static void main(String args[]) {
      float p = 5.5f;
      float q = 3.3f;
      System.out.println("The first number is:" + p);
      System.out.println("The second number is:" + q);
      float r = p + q;
      System.out.println("The addition is:" + r);
      r = p - q;
      System.out.println("The subtraction is:" + r);
      r = p * q;
      System.out.println("The multiplication is:" + r);
      r = p / q;
      System.out.println("The division is:" + r);
      r = p % q;
      System.out.println("The remainder is:" + r);
  }
}
```

In Listing 11, the arithmetic operations are performed on the float type variables. Figure 18 shows output of the `FloatApp` class:



▲ **Figure 18: Displaying Output of the FloatApp Class**

Apart from the arithmetic operations, you also need to understand the arithmetic assignment operator, which is discussed in the following section.

The Arithmetic Assignment Operator

The equal (=) symbol is known as the assignment operator. The assignment (=) operator is used to store or assign a value to a variable. For example, in expression, i=10, we are assigning a value 10 to the variable i. Remember that in expression i=10, we are not comparing the variable i with 10; rather we are assigning the value 10 to the i variable.

In Java, assignment operators can be used in two ways—simple and compound. In simple assignment, we use the equal (=) operator for assigning a value to a variable. The following code snippet shows an example of simple assignment:

```
x = 10;              //x stores 10
x = x + 10;          //x stores 20
x = x + 5 * 15;      //x stores 95
```

In compound assignment, we use the +=, -=, *=, /=, and %= operators. The following code snippet shows an example of compound assignment:

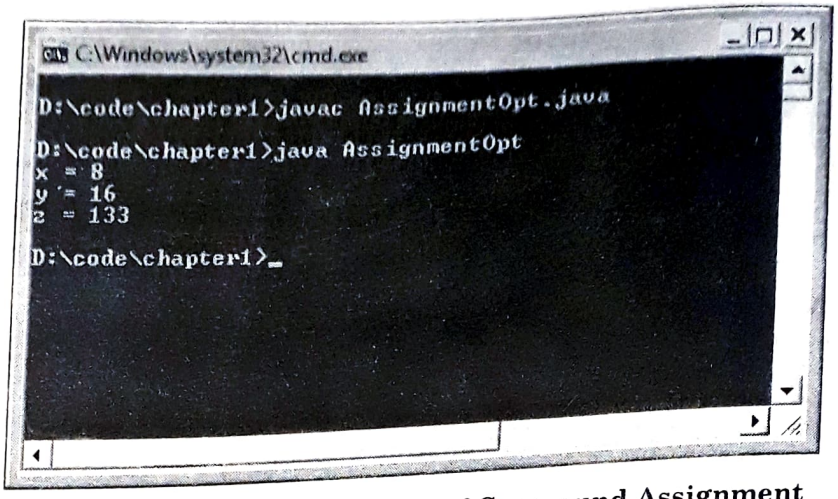
```
i = 10;       // i is initialized with the value 10
i += 15; // i is compound assigned with 25
i -= 5;       //i is compound assigned with 20
```

In the preceding code snippet, the variable i is assigned with the value 10. In the expression i+=15, the value 15 is added to the value already contained in the variable i and the total value is reassigned to i, so i will contain 25. In the expression i-=5, the value 5 is subtracted from the value already contained in the variable i and then the resultant value is reassigned to i, so in this case the variable i will contain 20. Listing 12 shows an example of compound assignment (you can find the AssignmentOpt.java file on the CD in the code\chapter1 folder):

▶ **Listing 12: Example of Compound Assignment**

```
public class AssignmentOpt {
  public static void main(String args[]) {
      int x = 3;
      int y = 4;
      int z = 5;
      x += 5;
      y *= 4;
      z += x * y;
      System.out.println("x = " + x);
      System.out.println("y = " + y);
      System.out.println("z = " + z);
  }
}
```

In Listing 12, the `AssignmentOpt` class is defined. Within the `main()` method of the `AssignmentOpt` class, the three variables x, y, and z of int type store the values 3, 4, and 5, respectively. The expressions x+=5, y*=4, and z += x*y are performing compound assignment operations that reassign the values 8, 16, and 133 to the variables x, y, and z, respectively. Figure 19 shows the final values stored in the variables x, y, and z, after evaluating the expressions using the assignment operator:



▲ Figure 19: Displaying Result of Compound Assignment

Now, after understanding the implementation of the arithmetic operators, let's discuss the relational operators.

➡ The Relational Operators

Operators that are used to determine the relationship between two operands are known as relational operators. The expressions that contain relational operators always return a boolean value, which is either true or false. There are six relational operators that can be used to compare any value of integers, floats, or characters. Table 7 lists relational operators:

Table 7: Relational Operators

Relational operators	Description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

All relational operators are binary operators. The relational operators can be used as given in the following code snippet:

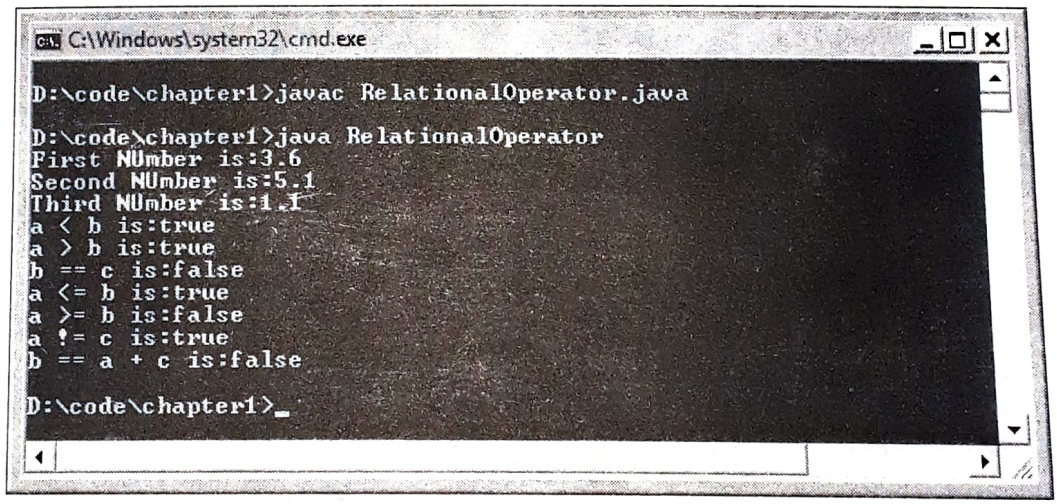
```
int x = 5;
int y = 8;
int z = 10;
boolean valid = x >= y;      // false.
boolean valid1 = x >= y && y >= z; // false.
```

In the preceding code snippet, the expression x >= y returns a false value, which is stored in the valid variable of boolean type. Similarly, the expression x >= y && y >= z is evaluated as (x >= y) && (y >= z), which returns a false value. This false value is stored in the valid1 variable of boolean type. Listing 13 shows the use relational operators (you can find the RelationalOperator.java file on the CD in the code\chapter1 folder):

▶ **Listing 13: Usage of Relational Operators**

```
/* Program implementing the Relational Operators */
class RelationalOperator {
  static int value1 = 10;
  static int value2 = 8;
  public static void main(String args[]) {
      float a = 3.6f;
      float b = 5.1f;
      float c = 1.1f;
      System.out.println("First Number is:" +a);
      System.out.println("Second Number is:" +b);
      System.out.println("Third Number is:" +c);
      System.out.println("a < b is:" + (a<b));
      System.out.println("a > b is:" + (a<b));
      System.out.println("b == c is:" + (b==c));
      System.out.println("a <= b is:" + (a<=b));
      System.out.println("a >= b is:" + (a>=b));
      System.out.println("a != c is:" + (a!=c));
      System.out.println("b == a + c is:" + (b == a + c));
  }
}
```

Listing 13 declares the `RelationalOperator` class. Within the `main()` method of the `RelationalOperator` class, three float type variables a, b, and c are initialized with the values 3.6f, 5.1f, and 1.1f, respectively. Next, the values of the a, b, and c variables are compared by using the relational operators ==, !=, <, and >, and then the result is displayed accordingly. Figure 20 shows the output of Listing 13:



▲ **Figure 20: Usage of Relational Operators**

From Figure 20, it is clear that the values contained in the variables a, b, and c are not equal and also depending upon the comparison, the boolean value is displayed.

Now after understanding the relational operators, let's study the boolean logical operators.

The Boolean Logical Operators

These operators operate only on the boolean operands. All of the binary logical operators combine two boolean values and provide the resultant boolean value. Table 8 lists the boolean logical operators:

Table 8: The Boolean Logical Operators

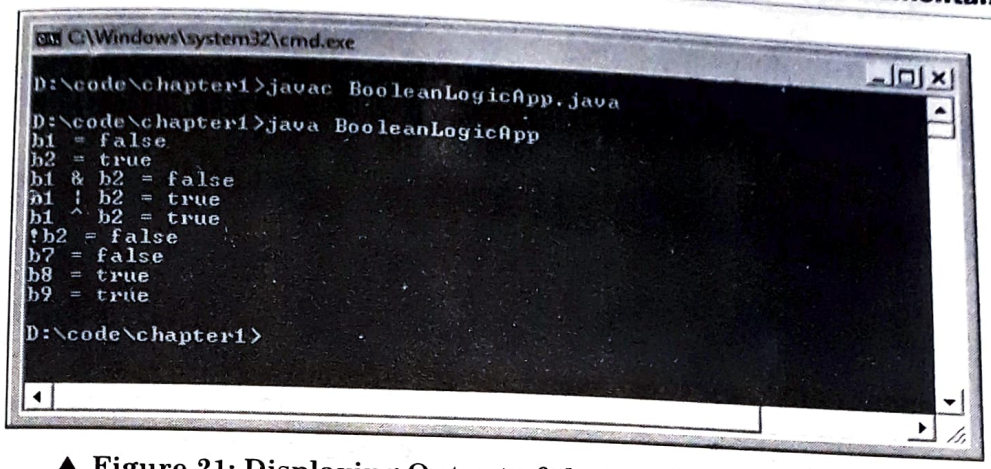
Operator	Its Meaning
&	Logical AND
|	Logical OR
^	Logical XOR (Exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical Unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not Equal to

In order to understand implementation of the boolean logical operators provided in Table 8, let's create a simple program. Listing 14 shows the implementation of the boolean logical operators (you can find the BooleanLogicApp.java file on the CD in the code\chapter1 folder):

▶ **Listing 14: The BooleanLogicApp.java File**

```
/* Program implementing the boolean logical operators */
class BooleanLogicApp {
 public static void main(String args[]) {
     boolean b1 = false;
     boolean b2 = true;
     boolean b3 = b1 & b2;
     boolean b4 = b1 | b2;
     boolean b5 = b1 ^ b2;
     boolean b6 = !b2;
     System.out.println("b1 = " + b1);      // false
     System.out.println("b2 = " + b2);      // true
     System.out.println("b1 & b2 = " + b3);    // false
     System.out.println("b1 | b2 = " + b4);    // true
     System.out.println("b1 ^ b2 = " + b5);    // true
     System.out.println("!b2 = " + b6);        //false
     int i = 10;
     boolean b7 = 6 == 4 & i < 4;          // false
     boolean b8 = b3 ^ b2;              // true
     boolean b9 = b4 | b1 & b2;         // true
     System.out.println("b7 = " + b7);
     System.out.println("b8 = " + b8);
     System.out.println("b9 = " + b9);
 }
}
```

In Listing 14, the boolean logical operators, such as !, ^, &, and ! are used. Depending upon the boolean logical comparison, the boolean value is returned and displayed as output. Figure 21 shows output of the BooleanLogicApp class:



▲ **Figure 21: Displaying Output of the BooleanLogicApp Class**

Figure 21 shows the boolean values returned on the basis of the boolean logical operators.

TEST YOUR KNOWLEDGE

Q11. Write a program to find the largest value among a, b, and c by using the && operator.

Ans.

```
/* Program to demonstrate how to find the largest value using the &&
operator */
class TestYourKnowledge11
{
  public static void main (String args[ ])
  {
      int a=11,b=8,c=18;
      if((a>b)&&(a>c))
      System.out.println("BIG="+a);
      else if((b>a)&&(b>c))
      System.out.println("BIG="+b);
      else
      System.out.println("BIG="+c);
  }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge11.java
D:\code\chapter1>java TestYourKnowledge11
```

Output:

```
BIG=18
```

The Increment and Decrement Operators

The increment and decrement operators can be used in two ways: either before or after an operand. Depending on the placement of these operators, they can be classified as prefix or postfix. The difference between prefix and postfix notations is that in the prefix notation, the value of the operand is incremented or decremented before assigning it to another variable, while in the postfix notation, the value of the operand is incremented or decremented after assigning it to another variable.

Consider the following code snippet:

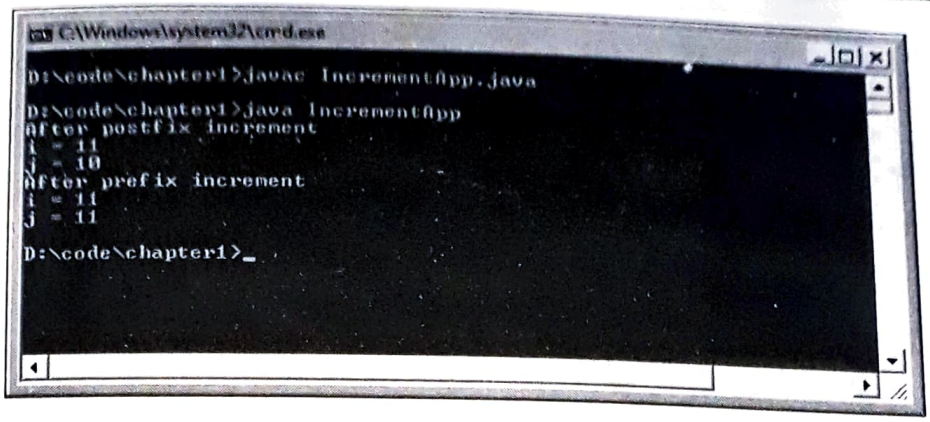
```
int x=10;
int y=20;
++x;
  //Prefix notation of increment operator
--y;
  // Prefix notation of decrement operator
x++;
  //Postfix notation of increment operator
y--;
  // Postfix notation of decrement operator
```

In the preceding code snippet, ++x is the prefix notation of the increment operator and --y is the prefix notation of decrement operator. On the other hand, x++ is the postfix notation of the increment operator and y-- is the postfix notation of the decrement operator. Listing 15 shows how to use the prefix and postfix notations of the increment (++) operator (you can find the IncrementApp.java file on the CD in the code\chapter1 folder):

▶ Listing 15: Usage of the Increment (++) Operator

```
/* Program implementing the Increment operator */
class IncrementApp {
 public static void main(String args[]) {
     int i, j, k;
     i = 10;
     j = i++;   // using postfix increment operator j = 10
     System.out.println("After postfix increment");
     System.out.println("i = " + i);
     System.out.println("j = " + j);
     i = 10;
     j = 0;
     j = ++i;   // using prefix increment operator j = 11
     System.out.println("After prefix increment");
     System.out.println("i = " + i);
     System.out.println("j = " + j);
 }
}
```

In Listing 15, the `IncrementApp` class is defined. Within its `main()` method, the two variables i and j of int type are declared. The variable `i` is initialized with number 10. The number that is stored in the i variable is incremented by one when the `i` variable is post-incremented (i++). It means that the value of the i variable will be 11; however, the variable `j` still stores the number 10, because in the postfix notation, the value of the operand is incremented or decremented after assigning it to another variable. Next, the i and j variables are reassigned numbers 10 and 0, respectively. When the i variable is pre-incremented (++i), its value is incremented by one. It means that the value of the i variable will be 11; however, the variable j also stores the number 11, because in the prefix notation, the value of the operand is incremented or decremented before assigning it to another variable. Figure 22 displays the output of Listing 15:



▲ **Figure 22: Difference Between Post-Increment and Pre-Increment Operators**

TEST YOUR KNOWLEDGE

Q12. Write a program to demonstrate the implementation of prefix and postfix increment operators.

Ans.

```
//*Program to demonstrate the implementation of prefix and postfix
      increment operators*/
class TestYourKnowledge12
{
  public static void main(String args[])
  {
      int x =10;
      System.out.println("Prefix is:" + (++x));
      x=10;
      System.out.println("Postfix is:" + (x++));
  }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge12.java
D:\code\chapter1>java TestYourKnowledge12
```

Output:

```
Prefix is:11
Postfix is:10
```

The Conditional Operator

The conditional (?:) operator is a ternary operator that takes three operands. It works similar to an if-else statement. The conditional (?:) operator evaluates a boolean expression and gives the result based on this boolean expression. The conditional operator is constructed using a question mark (?) and a colon (:) symbol. The syntax to declare a conditional operator is:

```
operand1? operand2 : operand3
```

In this syntax, operand1 is any valid boolean expression, such as 10>5. If operand1 is true, then operand2 is evaluated; otherwise, operand3 is evaluated. For example:

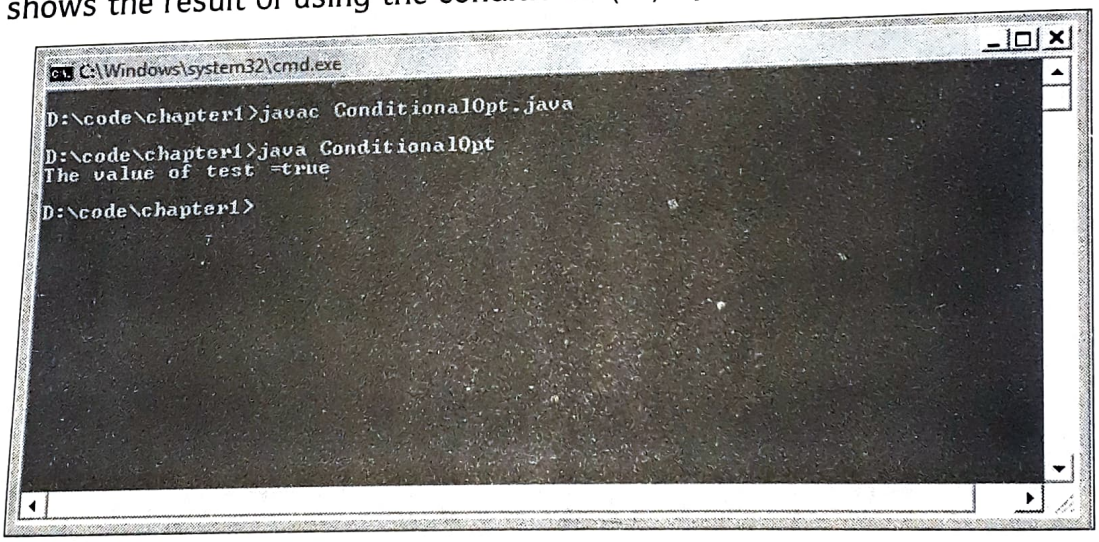
```
a = (x < y) ? x : y;
```

Listing 16 shows use of the conditional operator (you can find the ConditionalOpt.java file on the CD in the code\chapter1 folder):

▶ **Listing 16: Usage of the Conditional (?:) Operator**

```
/* Program implementing the Conditional Operator */
class ConditionalOpt
 {
  public static void main(String args[]) {
      int i=20, j=15 ;
      boolean test=false;
      test = i > j ?  true :  false;
      System.out.println("The value of test =" +test);
  }
}
```

In Listing 16, a class `ConditionalOpt` is defined. Within the `main()` method of the ConditionalOpt class, two variables i and j store the values 20 and 15, respectively. The boolean variable `test` stores the result of the conditional (?:) operator. In the expression i > j ? true : false, the value contained in the variable i is greater than that of the variable j; therefore, the?:operator returns true and the value is then assigned to the test variable. Figure 23 shows the result of using the conditional (?:) operator.



▲ **Figure 23: Usage of Conditional Operator**

TEST YOUR KNOWLEDGE

Q13. Write a program to demonstrate the use of ternary operator.

Ans.

```
/* Program using the ternary operator
*/
class TestYourKnowledge13
{
public static void main(String args[])
{
int x =3;
int y =7;
int z =0;
```

```
int res;
res = (x<y ? x : y)<z ? (x<y?x:y):z;
        System.out.println("The minimum number is:" + res);
}
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge13.java
D:\code\chapter1>java TestYourKnowledge13
```

Output:

```
The minimum number is:0
```

Q14. Write a program to find the largest value using the ternary operator (?).

Ans.

```
/* Program to demonstrate how to find the largest value using the ?
 operator */
class TestYourKnowledge14
{
    public static void main (String args []) {
        int a=3,b=7,c=13,big=0;
        big=(big>a)? big:a;
        big=(big>b)? big:b;
        big=(big>c)? big:c;
        System.out.println("BIG="+big);
    }
}
```

Execution:

```
D:\code\chapter1>javac TestYourKnowledge14.java
D:\code\chapter1>java TestYourKnowledge14
```

Output:

```
BIG=13
```

The Bitwise Operators

These operators perform on the individual bits (0 and 1) of their operands. They can be applied only on integer types that is long, int, short, and byte. Bitwise operators should be used where memory savings are needed, such as for system programming. Table 9 lists the bitwise operators:

Table 9: Bitwise Operators

Operator	Meaning
&	AND
|	OR
^	Exclusive OR

Table 9: Bitwise Operators

Operator	Meaning
>>	Right Shift
<<	Left Shift

Let's now discuss the operators listed in Table1.9.

The AND Operator

This operator produces 1 bit in case the corresponding bits in both the operands are 1; otherwise, it produces zero, as shown in Table 10:

Table 10: Outcomes with AND operator

A	B	A & B
0	0	0
1	0	0
0	1	0
1	1	1

The OR Operator

This operator produces 1 if one or both of the corresponding bits in its operands are 1, and produces 0 if both of the corresponding bits are 0. In other words, OR (|) operator returns one in all cases except in a situation where the corresponding bits of both operands are zero, as shown in Table 11:

Table 11: Outcomes with OR operator

A	B	A | B
0	0	0
1	0	1
0	1	1
1	1	1

The Exclusive OR Operator

This operator produces 1 where the corresponding bits in its operands are different and produces 0 if they are the same as shown in Table 12:

Table 12: Outcomes with Exclusive OR

A	B	A ^ B
0	0	0
1	0	1
0	1	1
1	1	0

Let's now discuss shift operators. Before that, we must understand what exactly does shifting mean. Shifting means moving bits left or right depending upon the kind of shift operator used. Here, one bit moves forward making space for the next bit to take its place.

The Right Shift Operator

This operator >> shifts the bits to the right by a specified number of positions. The number of positions upto which the shifting should occur depends on the value given after the operator. When shifting right >>, the lost bits are filled in with previous leftmost bits.

To understand right shift operator more clearly, let's see an example:

Consider 78 whose binary equivalent is:

```
00000000 00000000 00000000 01001110
```

By shifting it right >> 2 times, the output we get is:

```
00000000 00000000 00000000 00010011 ( = 19 )
```

The Left Shift Operator

This operator shifts the bits to the left by a specified number of positions. When you shift left <<, the void left behind by the shift is filled by zeros and the lost bits are filled in with previous rightmost bits.

To understand the left shift operator more clearly, let's see an example:

Consider 78 again whose binary equivalent is:

```
00000000 00000000 00000000 01001110
```

By shifting it left << 3 times, the output we get is:

```
00000000 00000000 00000010 01110000 ( = 624 )
```

Now let's create a simple program to demonstrate the bitwise logical operators.

Using the Bitwise Logical Operator

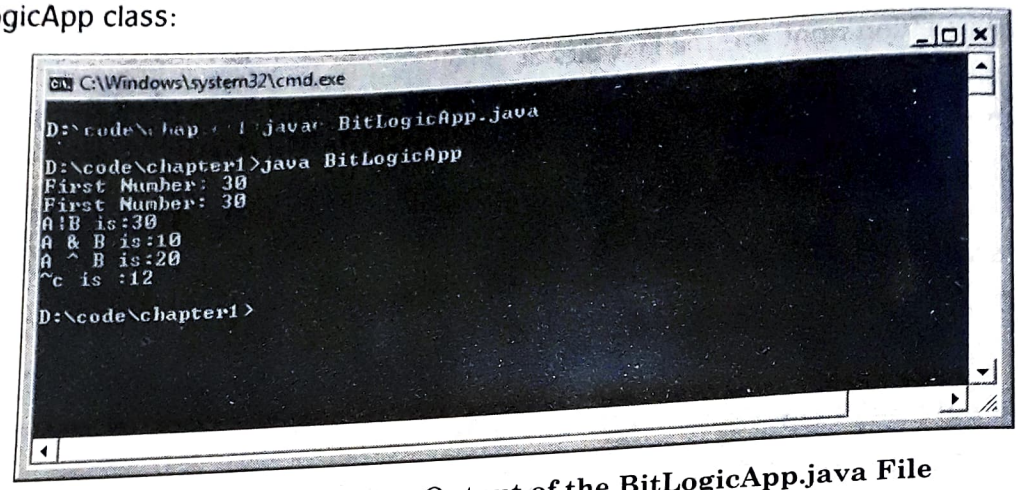
Here, let's create a simple program which explains the implementation of the bitwise logical operator. Listing 17 shows the use of bitwise logical operator (you can find the BitLogicApp.java file on the CD in the code\chapter1 folder):

▶ Listing 17: The BitLogicApp.java File

```
/* Program implementing bitwise logical operators */
class BitLogicApp {
  public static void main(String args[]) {
      int a = 30;
      int b = 10;
      int c = 3;
      System.out.println("First Number: "+ a);
      System.out.println("First Number: " + a);
      System.out.println("A|B is:" + (a | b));
      System.out.println("A & B is:" + (a & b));
      System.out.println("A ^ B is:" + (a ^ b));
```

```
        int d = ~c & 0x0f;
        System.out.println("~c is :" + d);
    }
}
```

In Listing 17, the bitwise logical operators, such as |, ^, and &. Figure 24 shows output of the BitLogicApp class:



▲ **Figure 24: Displaying Output of the BitLogicApp.java File**

Now, after understanding the implementation of the bitwise operators, let's discuss the implementation of the bitwise shift, i.e. left or right shift operators.

Using the Bitwise Shift Operators

Here, let's create a simple program to understand implementation of the bitwise shift operator. Listing 18 shows the usage of the left and right shift bitwise operator (you can find the BitShiftApp.java file on the CD in the code\chapter1 folder):

▶ **Listing 18: The BitShiftApp.java File**

```
/*Program implementing the bitwise shift operators */
class BitShiftApp
{
 public static void main(String args[])
 {
     int x=6;
     int y = -6;
     System.out.println("First Number:" +x);
     System.out.println("Second Number:" +y);
     System. out.println("X >> 2" +(x>>2));
     System.out.println("X << 1" +(x<<1));
     System.out.println("X >>> 1" +(x>>>1));
     System.out.println("y >> 1" +(y>>1));
     System.out.println("y >>> 2" +(y>>>2));
 }
}
```

In Listing 18, implementation of the left and right shift bitwise operators is shown in the BitShiftApp class. Figure 25 shows output of the BitShiftApp class, after compilation and execution:

▲ **Figure 25: Displaying Output after Using Bitwise Shift Operators**

LIBRARY
Women's Polytechnic
RANCHI

Special Operators

Here, we discuss the instanceof and dot operators. The instanceof operator is a binary operator that checks whether an object is of a particular type (here, type can be a class, interface, or an array). It is used for object reference variables only. The syntax to use the instanceof operator is:

```
<objectReference> instanceof <XxxType>
```

Here, the left-hand operand of the instanceof operator is object Reference, which must be an instance (or we can say an object) of a class while the right-hand operand (XxxType) can be a class, interface, or an array type. The instanceof operator returns true if the object mentioned on the left-hand side is an instance of the type that is mentioned at the right-hand side of this operator; otherwise, it (instanceof operator) returns false.

Apart from the instanceof operator, in Java we also use the dot (.) operator as shown in the following code snippet:

```
X x;
x.go();
```

In the preceding code snippet, the x instance of the X class is followed by a dot (.) operator to invoke the method, i.e. member of the X class.

Now after understanding all types of operators in detail, let's discuss the precedence of operators and their associativity.

Operator Precedence and Associativity

In Java, each operator has an associated precedence which helps in determining how an expression is evaluated if more than one operator is used. On the basis of operator precedence, the evaluation of an operator depends. In other words, the operators at the higher level of precedence are evaluated first and operators of same precedence are evaluated either from left to right or vice-versa. Such a property of an operator is known as associativity. Table 13 describes the operator precedence and its associativity:

Table 13: Operator Precedence and its Associativity

Order	Operator	Associativity
1	()	Left to Right
	[]	
2	(Unary Minus)	Right to Left
	++	
	--	
	!	
	~	
3	*	Left to Right
	/	
	%	
4	+	Left to Right
	(Subtraction)	
5	<<	Left to Right
	>>	
	>>>	
6	<	Left to Right
	<=	
	>	
	>=	
	instanceOf	
7	==	Left to Right
	!=	
8	&	Left to Right
9	^	Left to Right
10	|	Left to Right
11	&&	Left to Right
12	||	Left to Right
13	>:	Right to Left
14	=	Right to Left

Summary

The chapter discusses the need of OOP and its concepts, emergence of Java, and features of Java. The chapter also provides the code for a simple Java program and then discusses Java tokens, such as identifiers, literals, separators, and operators. The examples of legal identifiers are also provided in the chapter. Moreover, different types of operators, such as arithmetic, boolean, logical, and relational operators are discussed towards the end of the chapter. The Java programs are created to understand implementation of various types of operators in Java. Towards the end, operator precedence and its associativity are discussed.

Review Questions

Here let's answer the following types of questions:

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

True or False

State True or False for the following:

1. The addition (+), subtraction (-), and multiplication (*) operators are used only with integer numbers. **False**
2. In Java, the multiplication (*), division (/), and modulus (%) operators have the same level of precedence. **True**
3. Java does not contain the struct and union data type. **True**
4. Java supports operator overloading. **False**
5. In Java, the source file name must match with the class name in which the main method is invoked along with the .java extension. **True**
6. Java supports multi-threading and is platform-independent. **True**
7. The java compiler translates source code into the bytecode file. **True**
8. In Java, the opening and closing braces is similar to C++ class construct. **True**
9. Semicolon (;) is a separator, which terminates a statement in a Java program. **True**
10. The && operator is a relational operator. **False**
11. In the prefix form increment/decrement operator, the previous value is obtained for use in the expression, and then the operand is modified. **False**
12. While evaluating logical expression of the following type, both the expressions are evaluated: boolean expression1 && boolean expression2 **True**

Multiple Choice Questions

Answer the following choice question:

Q1. If you assign a fixed value to a variable, it is known as:

A. Constant
B. Identifier
C. Literal
D. Const

Ans. Option A is correct.

Q2. Which of the following is not a legal identifier?

A. a + b
B. final
C. #sal
D. _salary

Ans. Option D is correct.

Q3. Which of the following is not a keyword?

A. union
B. const
C. boolean
D. synchronized

Ans. Option A and B are correct.

Q4. Which of the following are correct?

A. int x = 16, x >> 2 =4;
B. int y = -8, y >>> 1 = -4;
C. int z = 16, z >> 2 = 4;
D. int p = -8, p << 1 =-4

Ans. Option A is correct.

Q5. What will be result of the 6+6 expression?

A. 1
B. 6
C. 12
D. 81

Ans. Option C is correct.

Q6. Consider the following code snippet:

```
int x, y;
x= 15;
y = 7;
```

What will be output of the x % y expression?

A. 1
B. 2
C. 0
D. 4

Ans. Option A is correct.

Q7. Which of the following serves as Java compiler?

A. javac
B. java
C. jar
D. jdb

Ans. Option A is correct.

Q8. Which of the following is the correct syntax of declaring an instance of a class?

A. Class_ name object_name
B. Class_name object_name;
C. Object_name class_name
D. Object_name class_name;

Ans. Option B is correct.

Q9. Which of the following is the correct order of precedence of the unary minus operator?

A. 4
B. 3
C. 2
D. 1

Ans. Option D is correct.

Q10. Which of the following is not the Java edition?

A. Java EE
B. Java SE
C. Java ME
D. Java RE

Ans. Option D is correct.

Q11. **Which of the following is not a feature of Java?**

A. Platform Independent
B. Robust
C. Dynamic and Multi-Threaded
D. Difficult to understand

Ans. Option D is correct.

Q12. **Which of the following is not a bitwise operator?**

A. Exclusive OR
B. instanceof
C. Right Shift
D. Left Shift

Ans. Option B is correct.

Short Answer Questions

Q1. **What do you mean by OOP?**

Ans. OOP is a technique that allows you to implement that famous programming dictum: "divide and conquer". The idea behind OOP is to encapsulate private data and methods into objects, thereby making each object semiautonomous, and enclose the data and methods in a way that prevents them from cluttering the general namespace.

Q2. **Explain the role of compiler and interpreter in Java.**

Ans. In all programming languages, a compiler translates source code into machine code for a specific computer. In Java, a compiler translates the Java source code into an intermediate code known as bytecode. In order to run a Java program on a different computer, you need a Java interpreter for Java bytecode. Although, depending upon the type of operating system in use, the Java bytecode interpreter varies, but once a computer has an interpreter, it can run a Java bytecode program.

Q3. **What do you mean by inheritance?**

Ans. Inheritance is one of the OOP concepts by which a class acquires the properties of another class. The concept of inheritance supports hierarchical classification. Without inheritance, you need to define all the characteristics of each object explicitly.

Q4. **What is the role of JVM in Java?**

Ans. In all the programming languages, a compiler translates source code into machine code for a specific computer. In Java, a compiler translates the Java source code into an intermediate code known as bytecode for a virtual machine, known as Java Virtual Machine (JVM). Bytecode is the machine language for JVM and is not machine specific. Instead, the machine specific code is generated by the Java interpreter, which serves as a mediator between virtual machine and real machine.

Q5. **Java is a platform-independent language. Justify**

Ans. Platform independency is one of the most important features of Java as it is the first programming language that is not bound to any specific operating system. In other words, Java programs can be executed anywhere on any system. No other programming language, before the evolution of Java, was platform independent and therefore, it can be said that Java is a revolutionary technology. Apart from cross-platform in source form, Java is also platform-independent in complied binary form.

Q6. List all the operators used in Java.

Ans. The operators in Java are as follows:

- Arithmetic
- Relational
- Boolean logical
- Increment and decrement
- Conditional
- Bitwise
- Special operators, such as instanceof and dot

Q7. What are the rules which you must keep in mind to declare a legal identifier?

Ans. An identifier is a symbolic name that refers to the name of variables, methods, classes, packages, or interfaces in a Java program. Legal identifiers are identifiers that conform to the rules that Java compiler uses to determine whether or not the name is legal. In Java, a legal identifier:

- Must start with an underscore (_), a letter, or a dollar sign ($)
- May consist of a letter, a number, an underscore, and a dollar sign
- Does not allow spaces in between an identifier
- Cannot have a name that is similar to a Java reserved keyword
- Are case-sensitive, that is, the word salary, Salary, and SALARY are three different identifiers

Q8. What do you understand by operator precedence and its associativity?

Ans. In Java, each operator has an associated precedence that helps in determining how an expression is evaluated if more than one operator is used. On the basis of operator precedence, the evaluation of an operator depends. In other words, the operators at the higher level of precedence are evaluated first and operators of same precedence are evaluated either from left to right or vice-versa. Such a property of an operator is known as associativity.

Q9. What is the instanceof operator?

Ans. The instanceof operator is a binary operator that checks whether an object is of a particular type (here, type can be a class, interface, or an array). It is used for object reference variables only. The syntax to use the instanceof operator is:

```
<objectReference> instanceof <XxxType>
```

Q10. How can you say that Java is secure?

Ans. In today's world, everybody needs safety and security. Due to the threat of hackers, people feel unsafe while performing commerce over Internet. In order to overcome such fears, Java provides the safety and security features.

While using Java compatible browser, everyone can safely download Java applets by relaxing their minds from the fear of virtual infection or malicious intent. Since the Java programs are confined to the Java runtime environment, so they can access only that part of your computer which is required for program execution. The data outside Java runtime environment cannot be accessed by Java programs, and

therefore downloading a Java application through Internet will not harm your computer.

Q11. What do you understand by conditional operator?

Ans. The conditional (?:) operator is a ternary operator that takes three operands. It works similar to an if-else statement. The conditional (?:) operator evaluates a boolean expression and gives the result based on this boolean expression. The conditional operator is constructed using a question mark (?) and a colon (:) symbol. The syntax to declare a conditional operator is:

```
operand1? operand2 : operand3
```

Q12. Explain the two different types of increment and decrement operators.

Ans. The increment and decrement operators can be used in two ways, either before or after an operand. Depending on the placement of these operators, they can be classified as prefix or postfix. The difference between prefix and postfix notations is that in the prefix notation, the value of the operand is incremented or decremented before assigning it to another variable, while in the postfix notation, the value of the operand is incremented or decremented after assigning it to another variable.

Debugging Exercises

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

```
class Ques1
{
    public static void main(String args[])
    {
        System.out.println("Hello");
    }
}
```

Ans. Output of the preceding program is as follows:

Hello

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

```
class Ques2
{
 public static void main(String args[])
    {
    int i = 50;
    String str = (i > 50) ? "Hello" : (i < 50) ?
    "World" : "Java";
    System.out.println(str);
 }
}
```

Ans. Output of the preceding program is as follows:

Java

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

```
class Ques3
{
public static void main(String args[])
{
    boolean x = false;
    boolean y = true;
    boolean z = true;
    boolean a = (x = true) || (y = true) && (z = true);
    System.out.println(x + ", " + y +", "    +z);
}
}
```

Ans. Output of the preceding program is as follows:

true, true, true

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

```
class Ques4
{
public static void main(String args[])
{
    System.out.println(9 / 6 + 3 / 7 + 2.5);
}
}
```

Ans. Output of the preceding program is as follows:

3.5

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

```
class Ques5
{
public static void main(String args[])
{
    int i = 5;
    i *= ++i;
    System.out.println(i);
}
}
```

Ans. Output of the preceding program is as follows:

30

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

```
class Ques6
{
public static void main(String args[]) {
    System.out.println(16 % 3);
}
}
```

Ans. Output of the preceding program is as follows:

1

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

```
class Ques7
{
 public static void main(String args[])
 {
     System.out.println(16 == 3);
 }
}
```

Ans. Output of the preceding program is as follows:

false

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

```
class Ques8
{
 public static void main(String args[])
 {
     System.out.println(16 + 3);
 }
}
```

Ans. Output of the preceding program is as follows:

19

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

```
class Ques9
{
 public static void main(String args[])
 {
     int i = 5;
     i *= i++;
     System.out.println(i);
 }
}
```

Ans. Output of the preceding program is as follows:

25

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

```
class Ques10
{
 public static void main(String args[])
 {
     int i = 5;
     i *= i--;
     System.out.println(i);
 }
}
```

Ans. Output of the preceding program is as follows:

25

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

```
class Ques11
{
 public static void main(String args[])
 {
     int i = 5;
     i *= --i;
     System.out.println(i);
 }
}
```

Ans. Output of the preceding program is as follows:

20

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

```
class Ques12
{
 public static void main(String args[])
 {
     System.out.println(16 >>> 3);
 }
}
```

Ans. Output of the preceding program is as follows:

2