



Chapter 1

Introduction to Java

Learning Objectives

After completing this chapter, you will be able to:

- *Understand history, evolution, and features of Java.*
- *Understand object oriented concept.*
- *Understand Java compiler and interpreter.*
- *Install JDK.*
- *Write, compile, and run a Java program.*

INTRODUCTION

In this chapter, you will learn about fundamentals of Java and will get a brief idea of its history, evolution, and features. Also, you will learn how to install Java on your system and run Java programs. Moreover, you will gain knowledge about the concept of object oriented programming as well as its importance in developing Java programs.

HISTORY AND EVOLUTION OF JAVA

Java was developed by Sun Microsystems in USA in the year 1991. Initially, it was named as Oak by James Gosling, its inventor, but later in 1995, it was renamed to Java due to some legal issues. Before the development of Java, it was planned that it would be developed for the use of electronic devices and circuits, and the plan got successful. Later on, it was called the Green project that led to the invention of Java. Apart from its general use, it is considered as a leading web-based technology. When it was invented, nobody knew how popular it would be. This is the first object oriented language that is platform independent and can run on any platform. It is used in the corporate sector worldwide in the form of application. Though the base concept for the development of Java has been taken from C and C++, Java is extremely different from C or C++. It is true that Java changed the world in terms of technology.

FEATURES OF JAVA

Java became popular due to its advanced features such as platform independency, simplicity, security, and so on. Some of the major features of Java are described next.

Platform Independency

As you know that Java is a platform independent language, therefore, it can run on any operating system. For example, a Java application can run on Linux, Machintosh Windows and any other operating system. When Java runs on a system, it converts the source code into the byte code. This byte code is generated by Java Compiler with the help of JVM(Java Virtual Machine) and can work independently on any system. You will learn more about JVM later in this chapter. Due to its portability, an application can be created in Java on one platform and can be used on any other platform. For example, you can create a Java application in your laptop or notebook and again reuse it in your office on different platform.

Simplicity

The syntax of Java language is very simple compared to other languages. It is very much similar to C or C++ language. Every keyword in Java is meaningful, therefore, the action of a keyword can be easily identified by its name.

Double Stage System

Java offers you the facility to compile programs in double stage. In the first stage, the Java compiler converts the source code into the byte code and in the second stage, the Java interpreter converts that byte code into the machine code. Since the computer can understand the machine code, it executes the code and produces the output. This is called the Double Stage System and is illustrated in Figure 1-1.

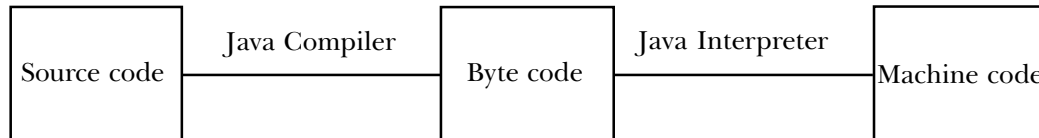


Figure 1-1 The Double Stage System in Java

Object Oriented

Java is completely object oriented programming language because it treats everything as an object. The basic concept of Java was taken from C and C++ language. C is not an object oriented programming language, but a structure based programming language. However, C++, which is an extension of C, is an object oriented programming language. Java is more independent than C or C++. You will know more about similarities and differences between Java, C, and C++ later in this chapter.

Security

Security is one of the most important issues in any programming language. If your application is not secure, your data is also not secure. Java is a security-based language. Because Java does not support the pointers like C++ does, so, the code is not able to access the memory directly. The internal system of Java starts verifying the code that tries to access its memory.

Multithreading

Java supports multithreading, which means that Java can handle multiple tasks in a single process. This is one of the most important features of Java. Also, it supports thread synchronization, which helps multiple threads work simultaneously in a synchronized way. You will learn more about multithreading in the later chapters.

Easy to Operate

Java is very user-friendly and easy to operate. There is no need to write a Java program in a particular environment. You can simply type the Java program in any text editor. For example, you can write a Java program in Notepad and after saving the program, you can execute the program in the command editor as you do in C or C++ programming language.

JAVA AND OBJECT ORIENTED PROGRAMMING

When talking about object oriented programming language, the name of Java comes first as it fulfills all requirements of a true object oriented programming language, such as it supports Data Abstraction, Encapsulation, Polymorphism, and Inheritance. These features are discussed in detail in the next section.

Data Abstraction

In terms of object oriented programming language, Data Abstraction means collecting different types of data in a group and explaining their common characteristics. It helps the developer face different behaviors of different types of data, and also helps the end user to control different data in a single entity. In addition, it helps to hide complexities of multiple data. Figure 1-2 illustrates the concept of Data Abstraction.

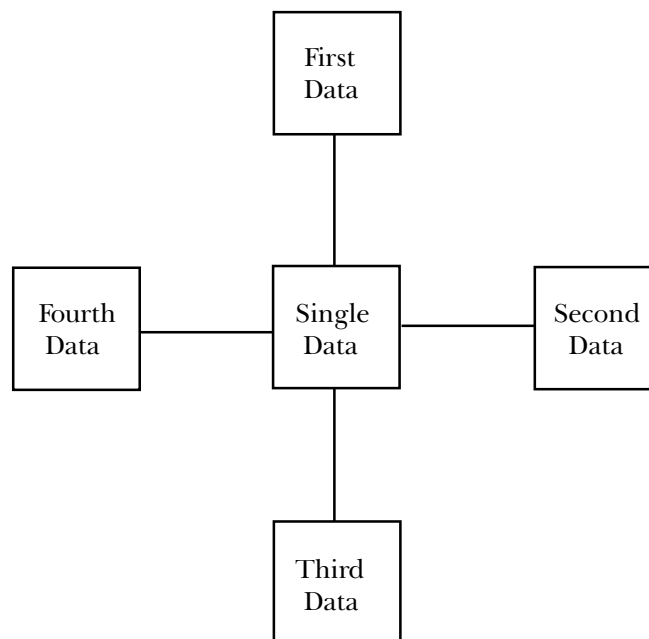


Figure 1-2 Data Abstraction

Encapsulation

This is another feature of object oriented programming language and is similar to Data Abstraction in terms of combining characteristics of data. In Data Abstraction, different types of data are collected in a single group and characteristics of all data are used, whereas in Encapsulation, methods and data are combined in a single unit. In other words, the method and data are encapsulated and they work as a single entity known as object, refer to Figure 1-3. This concept is also called data hiding. In this process, the data cannot be accessed by any external method or process, and only the methods that have been combined to work as a single entity can access it.



Figure 1-3 Encapsulation of data and method

Polymorphism

Polymorphism is a Greek word, wherein poly means many and morphism means form. In the object oriented concept, when a single operation plays multiple roles, it is called Polymorphism. For example, there is a function called **calculate(title)**. When this function is invoked with the argument **Add**, it will execute and give the result after addition. Again, if this function is invoked with the argument **Mul**, it will execute for multiplication and give the result. The same is true for subtraction and division. From this example, it is clear that a single function **calculate(title)** plays different roles when different arguments are passed to it. Generally, when the **Add** argument is passed to the function **calculate(title)** function, it converts the variable **title** to the variable **Add** and starts executing according to the condition specified in the **calculate(title)** function. Figure 1-4 illustrates the concept of Polymorphism.

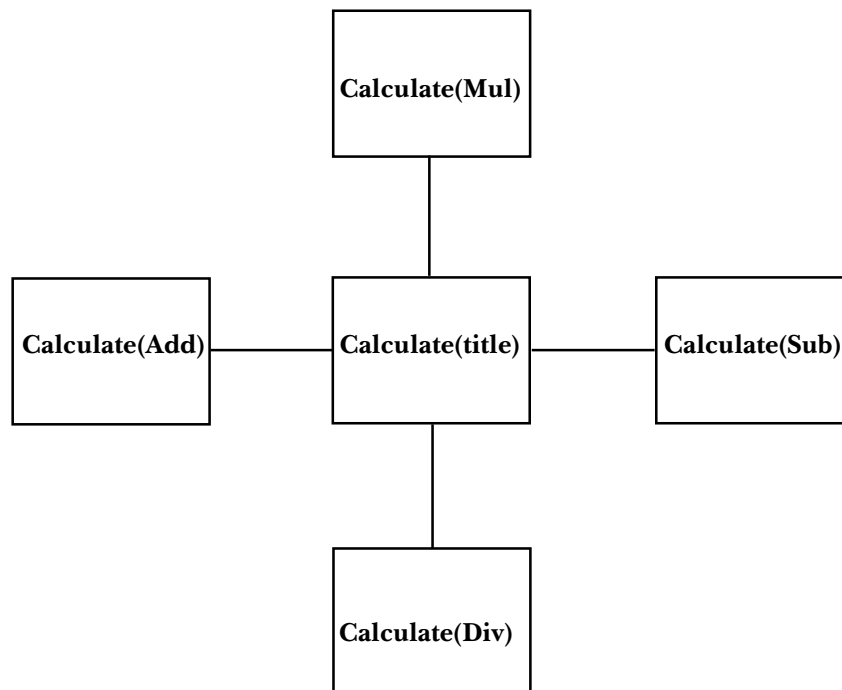


Figure 1-4 A single function with multiple operations

Inheritance

Inheritance is a key feature of object oriented programming. It gives you the benefit of reusing methods and properties of a class and hence, it reduces the lines of code in a Java program. When you create a class, you define some features in the form of properties and methods in it. Once the class is created, and later on you are required to create another class that has all features(properties and methods) of the existing class in addition with some new features(methods, properties). In this case, you do not need to create a separate class. You can do so by deriving a new class from the existing class and adding new features in the new class. By this way, you can avoid repeating same code.

Technically, inheritance is a technique of deriving new class from an existing class and reuse the features of the existing class in a new class. The derived class is also called sub class and the class from which you derive the new class is called super class or base class. Figure 1-5 will help you understand the concept of inheritance in a simple and easy way through an example.

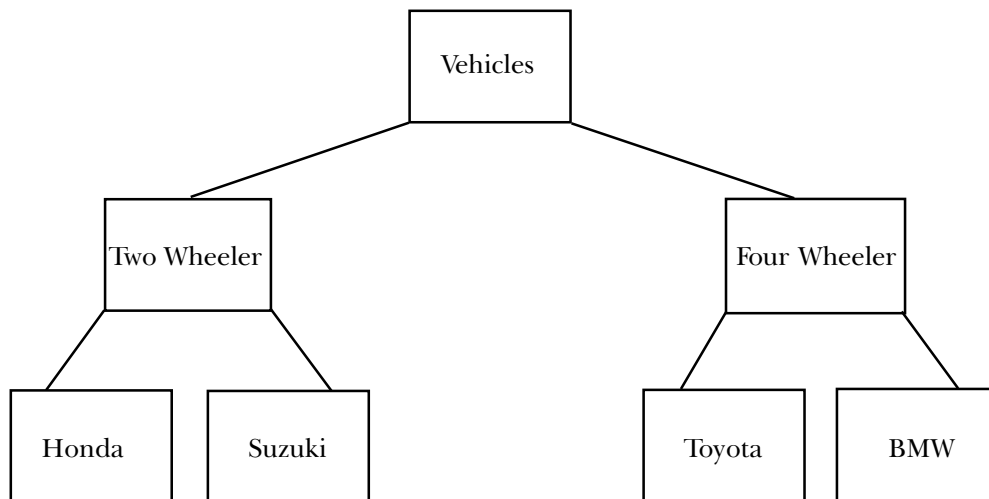


Figure 1-5 The Inheritance

In Java, inheritance is mainly of three types. These types are briefly explained next.

Single Inheritance

When a class is derived from only one super class, it is called Single Inheritance. Figure 1-6 illustrates the concept of Single Inheritance, wherein class B inherits the properties of class A.

Hierarchical Inheritance

When multiple classes are derived from a single class, it is called Hierarchical Inheritance. Figure 1-7 illustrates the concept of Hierarchical Inheritance, wherein class B and class C inherit the properties of class A.

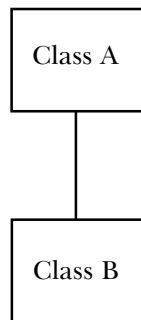


Figure 1-6 *Single Inheritance*

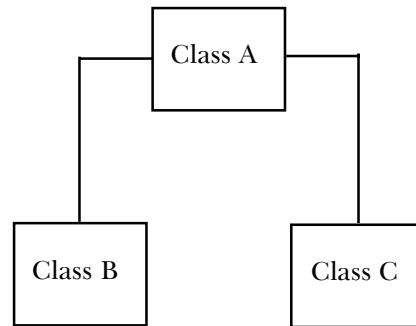


Figure 1-7 *Hierarchical Inheritance*

Multilevel Inheritance

When a class is derived from another derived class, it is called Multilevel Inheritance. Figure 1-8 shows Multilevel Inheritance, wherein class C inherits the properties of class B, which is a subclass of class A.

In object oriented programming, one more type of inheritance, called Multiple Inheritance is found. But this type of inheritance is not implemented in Java. In Multiple Inheritance, a class is derived from more than one super class. Figure 1-9 illustrates the concept of Multiple Inheritance. However, Java fulfills the requirement of Multiple Inheritance with the help of Interface. Interface is another important feature of object oriented programming and is discussed next.

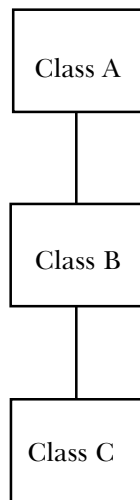


Figure 1-8 *Multilevel Inheritance*

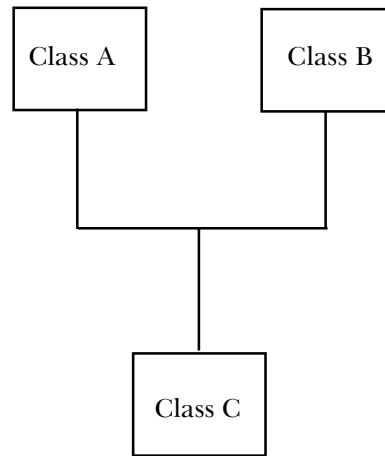


Figure 1-9 *Multiple Inheritance*

Interface

An interface is very much similar to a class with the only difference being that it does not include the code, which implements its own methods and properties. Interface can be used in place of Multiple Inheritance. It works like a template, which contains the required methods and properties that will be implemented only in the class, which is derived from this interface and another class. You can derive such a class from any number of interfaces. Figure 1-10 illustrates the concept of interface, wherein class C inherits the properties of interface A and interface B as well as class A.

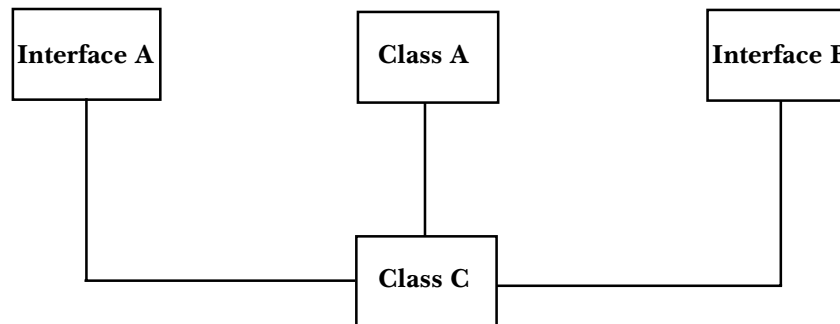


Figure 1-10 The Interface

Similarities and Differences between Java and C++

There are many similarities and differences between Java and C++. The major similarities between Java and C++ are as follows:

1. Both Java and C++ are object-oriented programming languages.
2. Both are case-sensitive.
3. Many keywords are same in Java and C++ such as **break**, **case**, **try-catch**, **throw**, and so on.
4. Both Java and C++ support **method overloading**.

The major differences between Java and C++ are as follows:

1. Java does not support Multiple Inheritance. You can use Interface instead of Multiple Inheritance in Java.
2. Java supports multithreading, whereas C++ does not.
3. Java does not support operator overloading.
4. Java does not support header files and pointers, whereas C++ does.

JAVA COMPILER AND INTERPRETER

When you run a Java program, it passes through two stages: compilation and interpretation. During compilation, the source code is converted into the byte code. Source code is nothing but a program that you have written in Java, whereas the byte code is a special type of code that is generated by the Java compiler. As it is not a machine specific code, it needs to be converted into machine level code, and this task is performed by Java Interpreter. The Java interpreter reads the byte code line-by-line and converts it into the machine level code. Now, the computer executes the machine level code.

Java Virtual Machine

You know that in a Java program, the source code is converted into the byte code, which is in turn converted to the machine code. In this process, the byte code is handled by a system called Java Virtual Machine (JVM). Therefore, the byte code is also sometimes called the Virtual Machine Code. JVM has a very powerful architecture. It is not simple machine or software, which can be installed on any system. Whenever you install JDK, it is automatically loaded into your computer memory and comes into action whenever you compile a Java program.

The diagrammatic representation of converting a Java program into a machine specific code is shown in Figure 1-11.

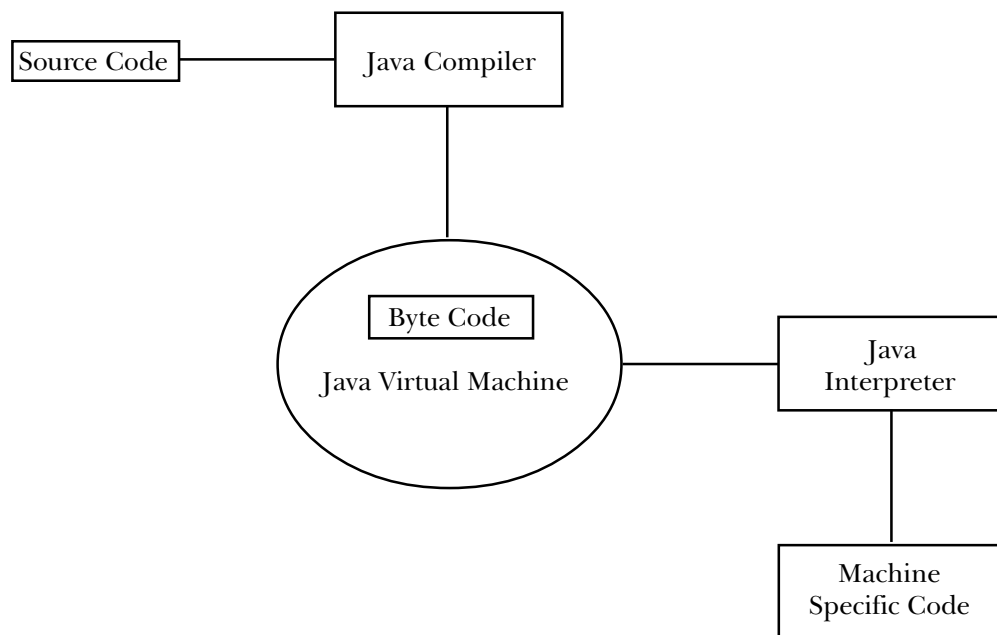


Figure 1-11 Converting Java program into machine specific code

INSTALLING JAVA DEVELOPMENT KIT

In this section, you will learn about the installation process of Java Development Kit(JDK) in your computer as well as the procedure to run a Java program. Installing JDK is a very easy process, and it takes few minutes to install it. You can download the Windows version of JDK from the Sun Microsystem's website to run on the Vista platform. As it is less than 100MB file, it takes few minutes to download, depending upon your internet speed and system configuration. You can try the following link to download the latest version of JDK:

<http://java.sun.com/javase/downloads/index.jsp>

Hardware and Software Requirements

To download the Windows version of JDK, you need to have any Windows or Vista platform with a memory of 64MB. In addition, you need to have a hard drive, IBM-compatible 486 system and a mouse.

Once the file is downloaded completely, you need to run the setup on your system. Initially, it will prompt you to accept the license agreement. Accept it and go ahead with the rest of installation process. During installation, leave the default settings of the setup intact. It will take a few minutes to complete the setup. Choose the **Finish** button to make sure that complete installation has been done. Now, you are ready to write and run Java programs.

Tools of JDK

The basic tools that will be installed when you install JDK on your system are discussed in the table given below:

javac	This is the Java compiler, which converts the Java source code into byte code.
java	This is the Java interpreter, which converts the byte code into machine specific code.
javadoc	This tool is used to create HTML documentation.
appletviewer	This tool interprets Java applet classes.
javah	This tool is used to write native methods.
javap	This tool is used as a disassembler of class files.
jdb	This is a Java debugger tool and is used to debug a Java program.
jar	This tool is used to manage Java Archive(JAR) files.

Table 1-1 Tools of JDK

JAVA STATEMENTS

Before writing and executing a Java program, you should know the basic keywords and syntaxes that are used in a simple Java program. Most of the keywords in Java are similar to C++. Some of the important keywords and syntaxes are discussed next.

Identifiers

Identifiers are basically the naming principles that help you identify the names of variables or classes while writing a Java program. Programmers use identifiers for their own convenience. To specify an identifier name, you must follow the rules given below:

1. The name should not begin with any digit.
2. It should be differentiated on the basis of upper case and lower case characters.
3. There is no limit to the length of an identifier.
4. The name can be a mixture of alphabets, digits, underscore, and dollar sign characters.
5. The name should be meaningful and short.
6. The reserved keywords of Java cannot be used as identifiers.

Following are some of the examples of identifiers:

area_of_rectangle, AreaOfRectangle, totalAmount, TOTAL_AMOUNT

Following are the reserved keywords of Java, which cannot be used as identifiers:

abstract	assert	boolean	break	byte	byvalue
case	cast	catch	char	class	const
continue	default	do	double	else	enum
enum	extends	final	finally	float	for
future	generic	goto	if	implements	import
inner	instanceof	int	interface	long	native
new	null	package	private	protected	public
return	short	static	strictfp	super	switch
synchronized	this	threadsafe	throw	throws	transient
try	var	void	volatile	while	

The Import Keyword

As the name suggests, this keyword is used to import packages, classes, or methods into your existing program. These packages or classes can be user-defined or in-built. For example, if you import a package into your existing program, you will not be able to access the features of all classes and methods of that package in your current program.

For example:

```
import java.lang.Math;
```

It indicates that the class **Math** of the package **lang** is the in-built package of Java.

The Class Keyword

The class is the base of object oriented programming language. It contains methods, objects, properties, variables, and many more. You will learn more about classes in the later chapters.

For example:

```
class Rectangle
```

This line indicates a **class** with the name **Rectangle**.

System.out.println() Statement

This statement is used to print the output to the next line of the command prompt. Here **println** indicates the member of the object **out**, whereas **System** indicates **class**.

For example:

```
System.out.println("Hello Java");
```

This statement will print the string **Hello Java** to the next line of the command prompt.

public static void main(String args[])

This statement is used in every Java program and is the main method of Java. The control will always reach to this statement.

public

This is an access specifier, any method with Public access specifiers will be accessible throughout the program.

static

The **static** keyword defines the method **main** as a separate method, which does not belong to any other method of class.

void

The keyword **void** indicates that the method **main** will not return any value.

main(String args[])

The **args[]** is an array of objects of the class **string**, which is the parameter of the method **main**.

Comments in Java

You can write comments in a Java program for your convenience. Java supports two types of comments that are also supported by C++. These are single line comment and multiple line comments.

Single Line Comment

A single line comment is used when you want to write a topic or a small comment in a Java program. An example of a single line comment is:

```
//This is a single line comment.
```

Multiple Line Comment

A multiple line comment is used when you want to write a long description or a small documentation of the program. An example of a multiple line comment is:

```
/*This is  
a multiple  
lines comment*/
```

Access Specifiers

Access specifiers are the keywords that can be used to declare a class or a method to make it accessible under different scopes within a class or a package. In Java, there are five types of access specifiers and they are explained next.

public

The **public** access specifier is declared when you want to make a method or any variable accessible in the entire class or package. Similarly, when you declare a class as **public**, it is accessible to the entire package.

For example:

```
public class Rectangle  
public calculate() {.....}
```

private

The **private** access specifier is declared when you want the fields of a class to be accessed only by that particular class and not by other classes.

For example:

```
private class Rectangle
private calculate() {.....}
```

protected

The **protected** access specifier is declared when you want to make the fields of a class accessible anywhere, except the base class in other packages.

For example:

```
protected class Rectangle
protected calculate() {.....}
```

private protected

The **private protected** access specifier is declared when you want to make the fields of a class accessible anywhere, except other classes of the same package and the base class in other packages.

For example:

```
private protected class Rectangle
private protected calculate() {.....}
```

friendly

The **friendly** is a default access specifier. If you do not specify any access specifier, the fields of a class are treated as **friendly**. The fields, declared as **friendly**, are accessible to all classes of the same package.

For example:

```
friendly class Rectangle
friendly calculate() {.....}
```

WRITING A JAVA PROGRAM

There are many editors available in the market to write and run a Java program. You can either use an editor from them or simply use a Notepad editor to write a Java program. In this section, you will learn how to write Java programs in a Notepad editor, and to compile and run them in command prompt. Given below is a simple Java program, which calculates the area of a rectangle.

```
// This program will calculate the area of a rectangle      1
import java.lang.Math;                                    2
class Rectangle                                           3
{                                                         4
    public static void main(String args[ ])              5
    {                                                     6
        double height=4.5;                               7
        double length=7;                                 8
        double area;                                     9
        area=length*height;                             10
        System.out.println("area of rectangle : " + area); 11
    }                                                     12
}                                                         13
```

Explanation

The line-by-line explanation of the above program is as follows:

Line 1

// This program will calculate the area of a rectangle

This line is used as a comment for understanding the purpose of the program.

Line 2

import java.lang.Math;

This line indicates that the class **Math** of the package **lang** has been imported into this program to access certain methods of the class **Math**. Here, **import** and **java** are the keywords, **lang** is the name of the package, and **Math** is the name of the class.

Line 3

class Rectangle

These lines indicate the creation of a class with the name **Rectangle**.

Line 4

{

This line indicate the opening of the curly brace, which contains a block of certain statements.

Line 7

double height=4.5;

This line indicates that a variable **height** of the data type **double** has been declared and initialized with the value **4.5**. You will learn about the data type in detail in later chapters.

Line 8

double length=7;

In this line, another variable **length** of the data type **double** and initialized with the value **7**.

Line 9

double area;

This line declares a variable with the name **area** of the data type **double**.

**Note**

If no value is specified in the program to initialize, it is initialized to the value 0 by default.

Line 10

area=length*height;

This line calculates the area of rectangle by multiplying the values of the variable **length** and **height** and then assigns the resultant value to the variable **area**.

Line 11

System.out.println("area of rectangle :" + area);

This line will print the area of the rectangle to the next line of the command prompt. The **System.out.println** is the method that is used to print the value in the next line of the command prompt as discussed earlier.

Line 12

}

This line indicate the closing of the curly brace and it contains a block of certain statements.

Compiling and Running a Java Program

Compiling and running a Java program is easier than a program written in other language. You can compile a Java program in any command editor. There are two ways to compile and run a Java program. The first way is to save the program in the bin directory directly, compile, and run it. The second way is to compile and run a program in your separate folder located anywhere in your drive by setting the path directory. Before compiling and running the above mentioned program, you need to know the two keywords that are used to compile and run a Java program. These keywords are discussed next.

The javac Command

The **javac** command is used to compile a Java program. It works only in the **bin** directory, where JDK is installed.

You can compile a Java program with the **javac** command in the following way:

```
javac Name_of_file with extension
```

Here, **javac** is a keyword and **Name_of_file** is the file name of the Java program with the extension *.java*. The most important point to be noted here is that the file name should exactly be the same as the class name. So, while naming a class or a file name, make sure both are same.

The java Command

The **java** command runs the Java program after compilation and then displays the output to the command prompt.

You can run a Java program with the **java** command in the following way:

```
java Name_of_file
```

Here, **java** is a keyword and **Name_of_file** is the file name of the Java program. Note that while executing the **java** command, you do not need to add extension of the Java file.

Steps to Write and Run a Program

You can write and run a Java program by following the steps given next.

Step 1

Open the Notepad editor from the **Start** menu and write the program in it line-by-line, as shown in Figure 1-12.

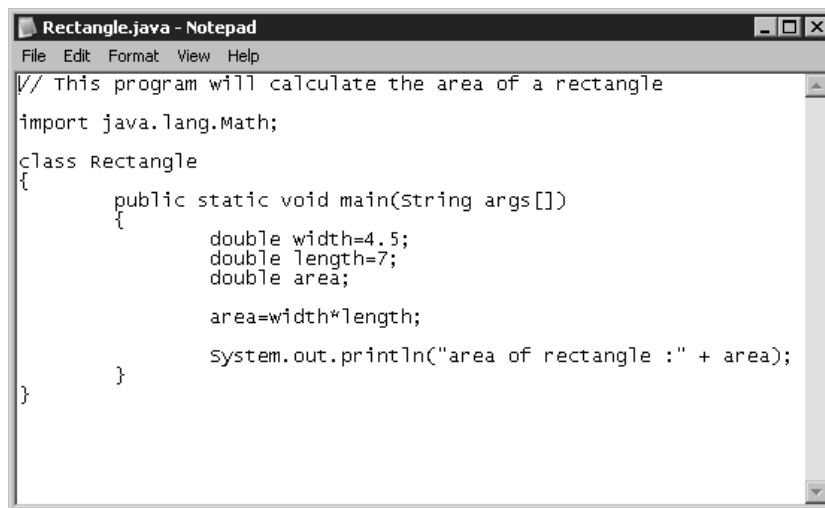


Figure 1-12 A Java program written in the Notepad editor

Step 2

Save the program with the file name **Rectangle.java** in the bin folder of the JDK directory. While saving the file, make sure the **Save as type** field displays **Text Documents(*.txt)** and the encoding field displays **ANSI**, as shown in Figure 1-13.

Step 3

Now, open the command editor from the **Start** menu. Enter the **bin** directory of the **Java** folder, where JDK is installed.

Step 4

Write the following **javac** command, give a space, type the name of file with extension, and then press ENTER.

```
Javac Rectangle.java
```

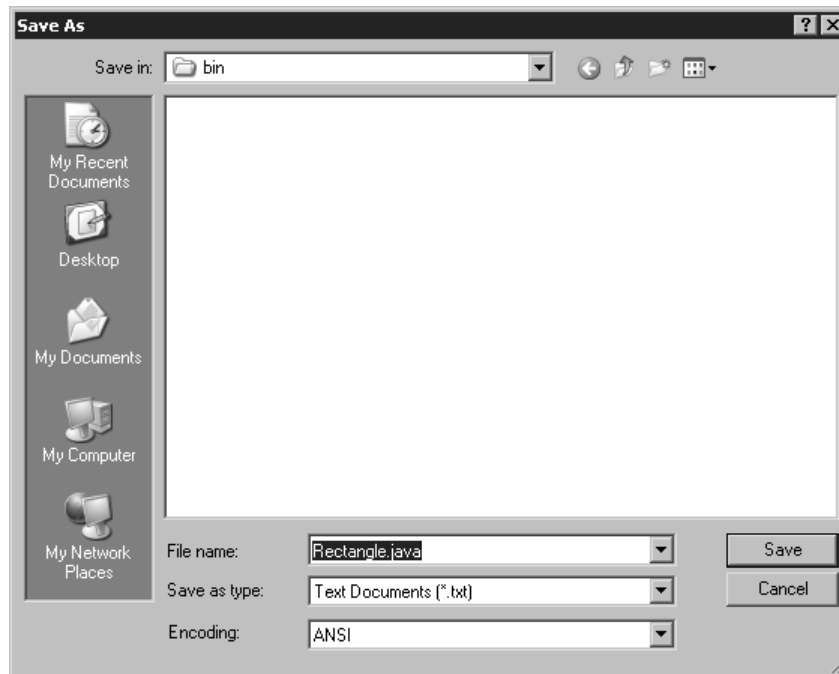
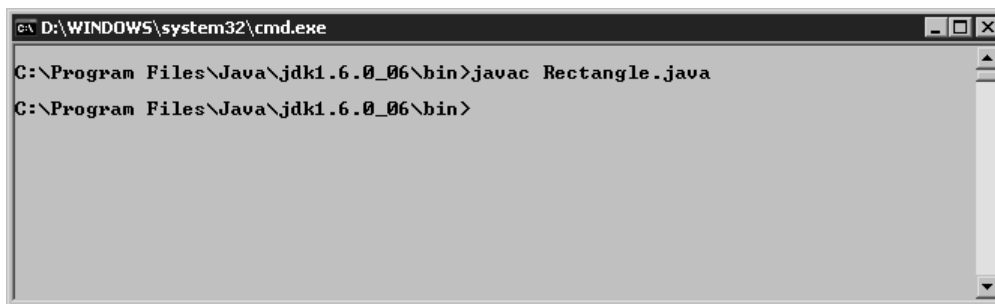


Figure 1-13 The Save As dialog box for saving the Java file

The screen capture given below illustrates the compilation process.

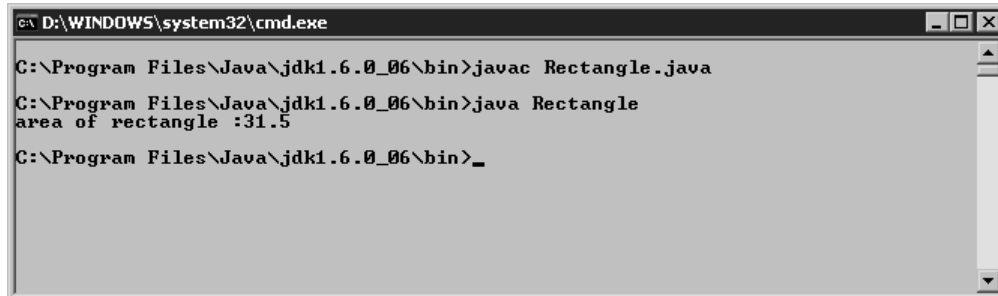


Step 5

Now, your program is compiled. If there is no error, you will not get any message on the command prompt after compilation. As soon as the program is compiled, a class file with the name *Rectangle.class* is generated in the bin directory. This is the class file, which generates after compilation. Now, type the following Java command at the command prompt, give one space, type the name of file without extension, and then press ENTER.

```
java Rectangle
```

Once you press ENTER, the output of the program will be displayed on the screen as follows:



```
C:\D:\WINDOWS\system32\cmd.exe

C:\Program Files\Java\jdk1.6.0_06\bin>javac Rectangle.java
C:\Program Files\Java\jdk1.6.0_06\bin>java Rectangle
area of rectangle :31.5
C:\Program Files\Java\jdk1.6.0_06\bin>_
```

SETTING THE PATH OF PROGRAMS FOLDER

It is recommended to set a path of the program folder, instead of compiling and running all Java programs in the bin directory. There are two methods for setting the path directory. First, you can set the path on a temporary basis, which means you can use this path till the command editor is open. Once you exit the command editor, you cannot use it again. Secondly, you can set the path permanently, which means the path will not be lost even after you shut down your computer. Both these methods are discussed next in detail.

Setting the Path on a Temporary Basis

The following steps are required to set the path directory of your Java project on a temporary basis:

Step 1

Create a folder anywhere in your hard drive containing all Java programs and name them as per your convenience; for example, **JavaProjects**.

Step 2

Open the command editor from the **Start** menu. Change the current directory and enter the drive and directory, where you have created the folder **JavaProjects**. For example, your folder is in the **D:** drive and you want to change the directory. You can do so by following the procedure given in the screen capture below:



```
C:\D:\WINDOWS\system32\CMD.exe

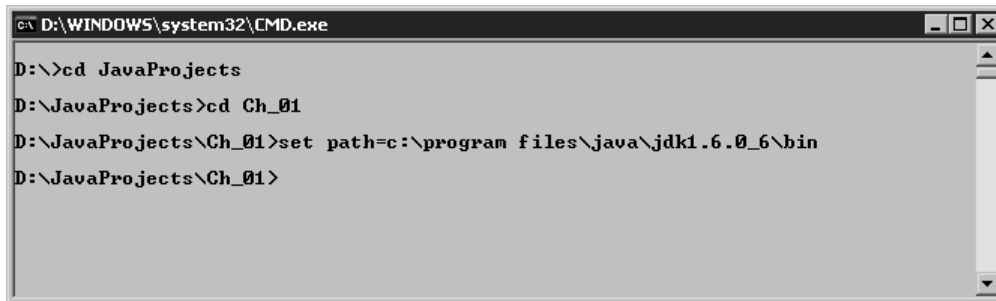
D:\>cd JavaProjects
D:\JavaProjects>cd Ch_01
D:\JavaProjects\Ch_01>_
```

Step 3

Write the following command at the command prompt:

path=The full path of bin directory of Java

Here, the **path** is the command. Write the full path of the bin directory of the Java folder after the assignment operator and then press ENTER, as shown next. The path will be set for the required directory.



```
C:\D:\WINDOWS\system32\CMD.exe
D:\>cd JavaProjects
D:\JavaProjects>cd Ch_01
D:\JavaProjects\Ch_01>set path=c:\program files\java\jdk1.6.0_6\bin
D:\JavaProjects\Ch_01>
```

Step 4

Now, you can compile and run all your programs in your project directory. There is no need to move your programs to the **bin** directory. Test your earlier program *Hello.java* by moving the file in your project directory. Repeat the same process for compiling and running the program that you had learned earlier. The program will be executed without any errors.

**Note**

Whenever you exit the command editor, the path setting is lost. So, you need to set the path each time you open the command editor.

Setting the Path Directory on a Permanent Basis

The following steps are required to set the path directory of your Java project on a permanent basis:

Step 1

Open **Control Panel** from the **Start** menu and then open the icon **System** by double-clicking on it; the **System Properties** dialog box will be displayed, as shown in Figure 1-14.

Step 2

Choose the **Advanced** tab and then choose the **Environment Variables** button from the dialog box; the **Environment Variables** dialog box will be displayed, as shown in Figure 1-15.

Step 3

Choose the **New** button in the **User variables** area of the **Environment Variables** dialog box; the **New User Variable** input box will be displayed, as shown in Figure 1-16. Enter the value **PATH** in the **Variable name** text box. Next, enter complete path of the **bin** directory of your JDK in the **Variable value** text box and choose the **OK** button.

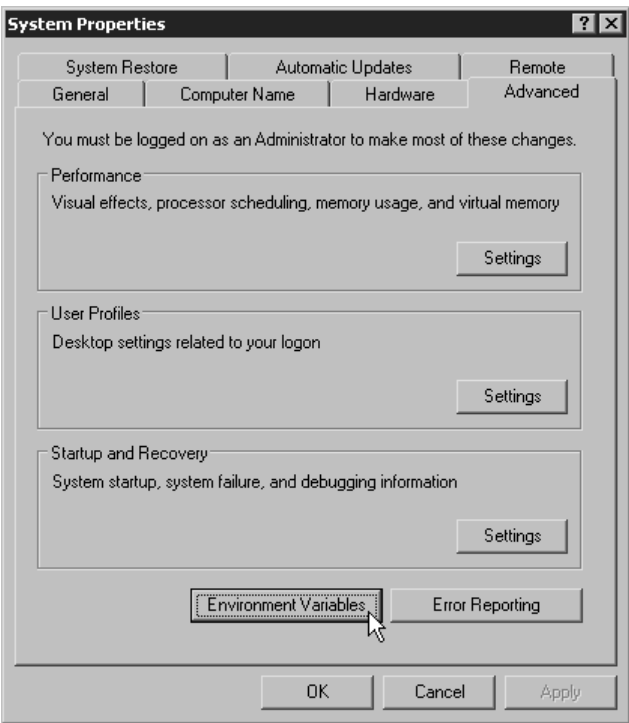


Figure 1-14 The *Advanced* tab in the *System Properties* dialog box

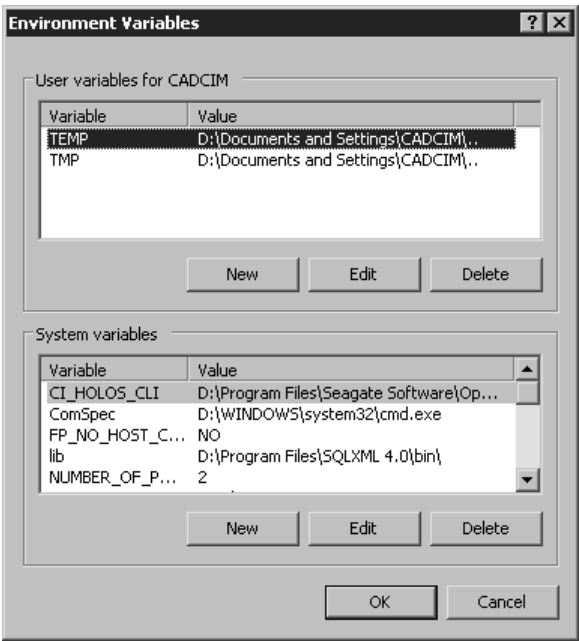


Figure 1-15 The *Environment Variables* dialog box

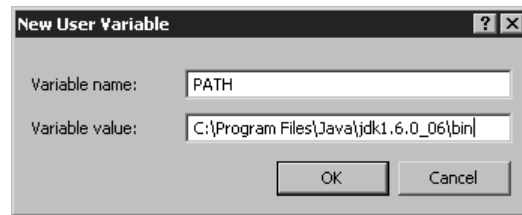


Figure 1-16 The New User Variable input box

Step 4

Choose the **New** button in the **System variables** area of the **Environment Variables** dialog box; the **New System Variable** input box will be displayed. Enter the same values in both the fields as you entered in the **New User Variable** input box, as shown in Figure 1-17. Choose the **OK** button. Again, choose the **OK** button of the **Environment Variables** dialog box and then choose the **OK** button in the **System Properties** dialog box.

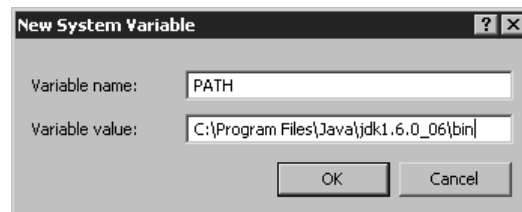


Figure 1-17 The New System Variable input box

Now, the path directory is set permanently and you can compile and run the program anywhere in your hard drive using the command editor.

JAVA API AND PACKAGES

Java API(Application Programming Interface) contains a number of classes and methods, which are grouped into different types of packages. Java API is used in various applications of Java. There are a variety of built-in packages in Java that can be used in different ways. Every existing class in Java belongs to a package. Some important packages that are widely used in Java programs are discussed next.

Java.lang

This is the most widely used package in Java, which provides the fundamental classes for programming. For example, **Integer**, **Float**, **Math**, **String**, **Thread**, and so on.

Java.io

This package is used to handle input and output files of Java programs. It contains classes like **Reader**, **Writer**, **Stream**, and so on.

Java.util

It contains miscellaneous utility classes involving data structures, string manipulation, time and date, and so on.

Java.net

This package is used for Java network programming, socket programming, and so on.

Java.awt

This package provides classes for the GUI(Graphical User Interface) based applications.

Creating Package

You can also create your own package by using the keyword **package** with a valid name. Packages created in this way are called user-defined package. An example of a package is given below:

```
package book;

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

In the above example, **package** is a keyword and **book** is the name of the package that contains a class with the name **JavaBook**. To import and use such type of packages in any other program, follow the syntax given next.

```
import book.*
```

Self-Evaluation Test

Answer the following questions and then compare them to those given at the end of this chapter:

1. Initially, Java language was given the name:
(a) Aok (b) Oak
(c) Ako (d) Oka
2. The project, which led to the invention of Java, is called:
(a) Light Project (b) Dark Project
(c) Red Project (d) Green Project
3. On which of the following platforms can Java run?
(a) Linux (b) Windows
(c) Only a (d) Both a and b
4. Which of the following is supported by Java?
(a) Data Abstraction (b) Encapsulation
(c) Polymorphism (d) All of these
5. Which of the following commands is used to compile a Java program?
(a) **java** (b) **javap**
(c) **javac** (d) **jvm**
6. Which of the following is used to set the path directory on a permanent basis?
(a) **Environment Variables** (b) **System Variables**
(c) **User Variables** (d) **Path Variables**
7. _____ helps in hiding the complexities of data.
8. In Encapsulation, methods and data are combined in a single unit. (T/F)
9. The **javac** command is used to interpret a Java program. (T/F)
10. Check the syntax error in the following program, which prints the message **Hello Java** to the next line of the command prompt.


```
Class Hello
{
    public static void main(String args[ ])
    {
        system.out.println("Hello Java");
    }
}
```

Review Questions

Answer the following questions:

- Which of the following inheritance is not supported by Java?
 - Multilevel Inheritance
 - Multiple Inheritance
 - Single Inheritance
 - Hierarchical Inheritance
- When multiple classes are derived from a single class, it is called
 - Single Inheritance
 - Multilevel Inheritance
 - Hierarchical Inheritance
 - Multiple Inheritance
- javadoc** is used for
 - Creating the Java documents
 - Creating the Java applets
 - Creating the HTML documentation
 - Creating the XML documentation
- Which one of the following is not a reserved keyword of Java?
 - assert**
 - catch**
 - implements**
 - write**
- Which of the following is supported by Java?
 - Multiple Inheritance
 - header files and pointers
 - Multithreading
 - Operator overloading
- The length of identifiers has no limit. (T/F)
- While executing the **java** command, there is no need to add the extension file(.java). (T/F)
- _____ editor can be used to write a Java program.

9. _____ prints the next line of the command prompt.
10. Write a program to calculate the area of a circle.

Answers to Self-Evaluation Test

1. b, 2. d, 3. d, 4. d, 5. c, 6. a, 7. Data Abstraction, 8. T, 9. F, 10. Error in system, s will be written in upper case.