

Chapter 1

Introduction to Java

Learning Objectives

After completing this chapter, you will be able to:

- *Understand history, evolution, and features of Java*
- *Understand the concept of OOPS*
- *Understand Java compiler and interpreter*
- *Install Java development kit*
- *Write, compile, set the path, and run your first Java program*
- *Install NetBeans IDE*
- *Write, build, and run Java program in NetBeans IDE*

INTRODUCTION

This chapter introduces you to Java programming language and allows you to write your first program in Java. In this chapter, you will get a brief idea about history, evolution, and features of the language. Also, you will learn how to install Java and Net Beans IDE (Integrated Development Environment) 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.



Note

1. *An Integrated Development Environment (IDE) is a software application that provides comprehensive features to computer programmers for software development. It generally consists of a source code editor, build automation tools, and a debugger.*
2. *The examples in this book are tested on Windows platform which is itself written in Java.*

HISTORY AND EVOLUTION OF JAVA

The first version of Java was developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) in USA, in the year 1991. Initially, it was named as Oak by James Gosling, but later in 1995, it was renamed to Java. The first version of the application was 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 purpose use, it is considered as a leading web-based technology. When it was invented, nobody knew how popular it would be. It is the first object oriented programming language that is platform independent and can run on any platform. It allows the developers to follow “Write Once, Run Anywhere” (WORA) concept. Though the base syntax of Java has been taken from C and C++, still it is quite different from C or C++.

There are four primary version of Java available:

- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)
- Java Micro Edition (Java ME)
- JavaFX

The Java based applications used for computer software are developed on the Standard Edition of Java. Java based applications used for web servers are developed using the Enterprise Edition of Java. Applications for the multimedia platform are developed using JavaFX and the applications for mobile devices are developed on the Micro Edition of Java.

You can create two types of programs using the Java programming language: **applets** and **applications**. **Applets** are smaller pieces of programming codes intended for use on the web browsers. They are lighter applications, generally used to provide navigation enhancement or additional interactivity to the browser.

The other type of software you can create using Java is a **console application**. These applications are standalone programs meant to be run on your computer like any other program. IDEs like NetBeans comes with integrated console environment that you can use to run your Java programs.

**Note**

Unlike standalone applications, Java applets do not need any interpreter in order to execute.

FEATURES OF JAVA

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

Platform Independency

As you are already aware that Java is a platform independent language, implying that it can run on any operating system. For example, a Java application written on the Windows platform can run on Linux, Machintosh, 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 created in Java on one platform can be run on any other 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; as a result, you can easily identify the action of a keyword.

Double Stage System

Java offers you the facility to compile programs in two 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 a computer can understand the machine code, it executes the code and produces the output. This is called the Double Stage System.

Object Oriented

Java is completely an 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 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 to 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. It does not require any specific environment for writing a Java program. 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 it using the command window as you do in C or C++ programming language.

CONCEPT OF OBJECT ORIENTED PROGRAMMING

When talking about object oriented programming system, the name of Java comes first as it fulfills all requirements of a true object oriented programming system, such as it supports Data Abstraction, Encapsulation, Polymorphism, and Inheritance.

Classes, instances, and objects are integral part of a program, if it is created by using an object oriented programming language such as Java. A **class** can be defined as a template/blueprint that describes the behavior/state of different objects. **Objects** are the specific elements that exhibit the properties and behaviors defined by its class. Objects of the same type are said to be of same type or same class. Once you derive an object from a class, it becomes an **instance**. The actions that an object can take are called **methods**. Methods in Java are called procedures, methods, functions, or subprograms in other languages.

To understand the concept of classes and objects, let's say Vehicles is a class. Under Vehicles class, you have different types of vehicles such as Cars, Buses, Trucks, and so on. Cars, Buses, and Trucks are the objects under the Vehicles class.

Features of Object-Oriented Programming

There are certain features that have made object-oriented programming very popular. These features are discussed next.

Data Abstraction

In terms of object oriented programming language, Data Abstraction means showing only functionality and hiding implementation details. It helps in hiding the complexities of multiple data. Real world example of data abstraction is sending E-mail, wherein a user only composes a mail and sends it to another user without knowing the internal process running in background.

Encapsulation

This is another feature of object oriented programming language. In encapsulation, methods and data are combined or wrapped together 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 1-1. 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 1-1 Encapsulation of data and method

Polymorphism

Polymorphism is a Greek word, wherein poly means many and morph means form. So, polymorphism is “one name many forms”. In the object oriented concept, when a single operation plays multiple roles, it is called polymorphism.

The best example of polymorphism is method overloading. In method overloading, there can be more than one method with same name but having different number of arguments. For example, there can be a class called **calculate** with two methods **calculate_area(l, b)** and **calculate_area(s)**. These methods have the same name **calculate_area** provided they have different number of arguments. So when two values are passed by the user, the method having two arguments **calculate_area(l, b)** will be invoked. But if the user passes single value, the method having single argument **calculate_area(s)** will be invoked.

Inheritance

Inheritance is a key feature of object oriented programming. It gives you the benefit of reusing methods and properties of a class and therefore, 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 to some new features (methods and properties). In that 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 or child class and the class from which you derive the new class is called super class or base class or parent class. Figure 1-2 will help you understand the concept of inheritance in a simple and easy way through an illustration.

As shown in Figure 1-2, **Vehicles** is a base class and it has two sub classes: **Two Wheeler** and **Four Wheeler**. These two subclasses, **Two Wheeler** and **Four Wheeler**, further have two more subclasses each **Honda** and **Suzuki**, and **Toyota** and **BMW** respectively. So, **Two Wheeler** and **Four Wheeler** will be the base classes for its subclasses and **Vehicles** will be the base class for all the subclasses.

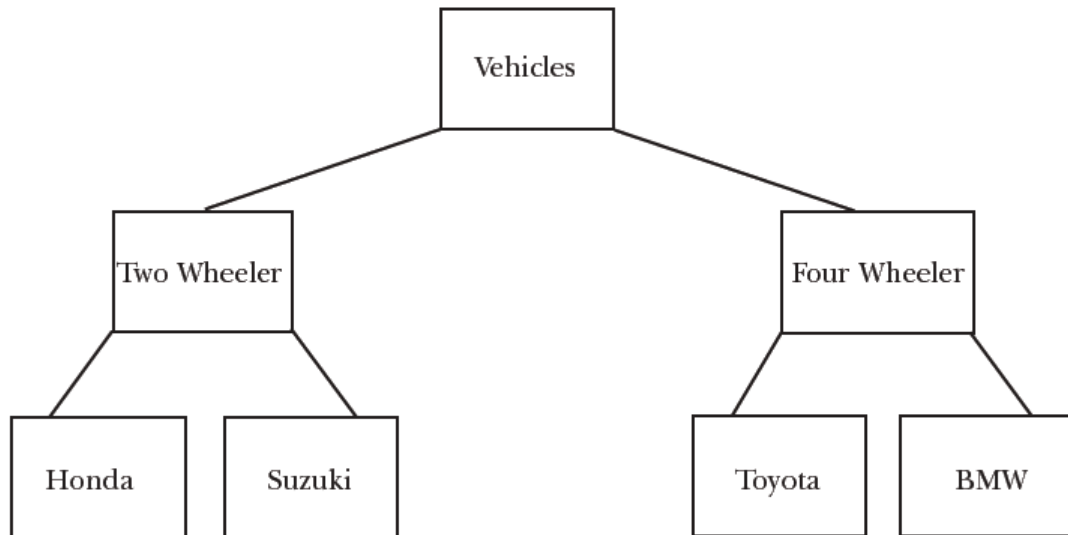


Figure 1-2 The concept of inheritance

Types of inheritance in Java

There are mainly three types of inheritance in java. These are as follows:

- a. Single Inheritance
- b. Multilevel Inheritance
- c. Hierarchical Inheritance

Single Inheritance

When a single class is derived from single super class, it is called single inheritance. Figure 1-3 illustrates the concept of single inheritance, wherein **class B** inherits the properties of **class A**.

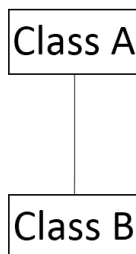


Figure 1-3 The concept of single inheritance

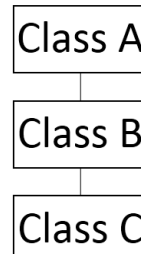


Figure 1-4 The concept of multilevel inheritance

Hierarchical Inheritance

When multiple classes are derived from a single base class, it is called hierarchical inheritance. Figure 1-5 illustrates the concept of hierarchical inheritance, wherein **class B** and **class C** inherit the properties of **class A**.

In object oriented programming, one more type of inheritance, called multiple inheritance is found. In multiple inheritance, a class is derived from more than one super class. The designers of Java considered multiple inheritance to be too complex and does not go well with the concept of keeping Java simple. Therefore, this type of inheritance is not implemented in Java. Figure 1-6 illustrates the concept of multiple inheritance.

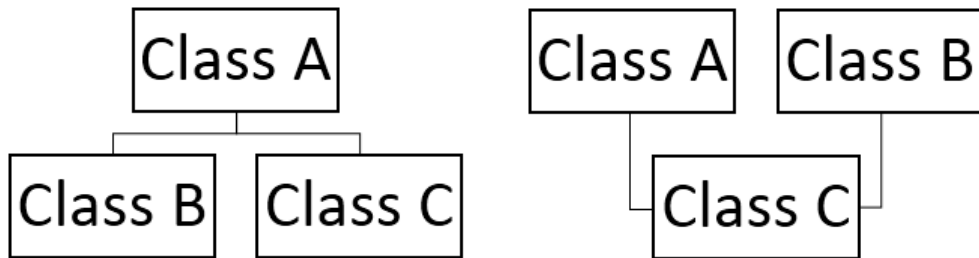


Figure 1-5 The concept of hierarchical inheritance

Figure 1-6 The concept of multiple inheritance

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.

Interface

An interface is a blueprint of a class. It is very much similar to the class but it contains only abstract method. An abstract method is a method that is declared but contains no implementation. The interface may also contain constants, method signatures, default methods, and static methods but it does not contain any constructor because it cannot be instantiated. Interfaces can only be implemented by classes or extended by other interfaces. A class inherits the abstract methods of an interface to implement it. You can use interfaces to achieve abstraction and multiple inheritance. You can also derive a class from any number of interfaces. Figure 1-7 illustrates the concept of interface, wherein **class C** inherits the properties of **interface A** and **interface B** as well as **class A**.

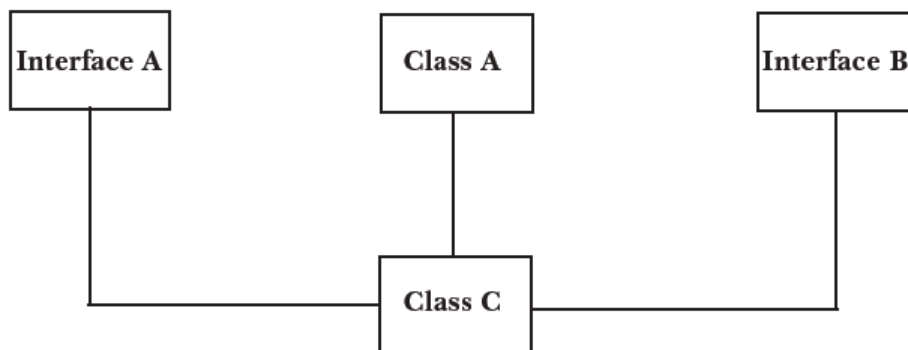


Figure 1-7 The concept of interface

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 an intermediate language by the compiler. Source code is a program written in Java and the intermediate code is a special type of code that is generated by the Java compiler. The intermediate code is known as Java byte-code or simple byte-code. As byte code 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.



Note

1. *Compiler is special purpose program that converts a high-level language (easy for people to write and to understand) such as Java program into a low-level language program (machine language).*

2. *Byte-code is a machine language for a virtual computer called Java Virtual Machine (JVM). Each computer platform has its own program to execute the byte-code instructions.*

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. Byte-code is not the machine language for any type of computer. In fact, it is a machine language for the fictitious computer called Java Virtual Machine or JVM. The term JVM is used to refer to the software that acts like a fictitious computer.

JVM has a very powerful architecture. 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 1-8.

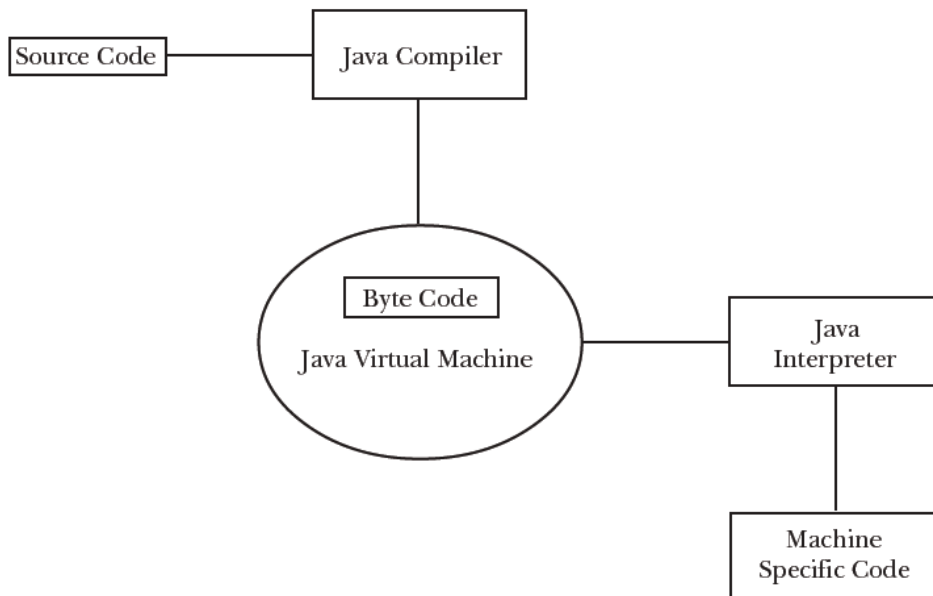


Figure 1-8 Converting Java program into machine specific code

INSTALLING JAVA DEVELOPMENT KIT

In this section, you will learn about the installation process of Java SE 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 Oracle website to run it on the Windows platform. You can try the following link to download the latest version of JDK:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Initially, you will be prompted to accept the license agreement, select the **Accept License Agreement** radio button and then click the `jdk-8u111-windows-x64.exe` corresponding to the Windows x64 entry. Save the file on your hard drive. If you have a 32-bit machine, click on the `jdk-8u111-windows-i586.exe` link corresponding to the Windows x86 entry. Once the file is downloaded completely, you need to run the setup on your system. 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 1-1.

Table 1-1 List of JDK tools

Tool	Description
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.

JAVA STATEMENTS

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

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. Packages can be of two types.

- a. Built-in packages
- b. User defined packages

Built-in Packages

Built-in packages are predefined in the Java library. 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 built-in 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.

User-defined Packages

You can also create your own package by using the **package** keyword 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 this 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.*
```

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 do not import a package into your existing program, you will not be able to access the features of all classes and methods of the package in your current program. To access the features of all classes and methods of the package, you need to import the package into your program.

For example:

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

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

You can also import the package as follows:

```
import java.lang.*;
```

Here * means all the classes in the package. It imports all the classes of the **lang** package.

The class Keyword

The class is the base of object oriented programming language. It contains methods, objects, properties, variables, and many more. Any valid name of class can be used to define a class prefix by a **class** keyword.

For example:

```
class JavaBook
```

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

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 **out** object, 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.

**Note**

Instead of keyword ***println***, you can use keyword ***print***. Both are same, but the only difference is, ***println*** shows the output in next line whereas ***print*** shows the output in the same line.

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 four 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 of a class accessible to any class in the Java program. These classes may be in the same package or in another package. The public access specifier achieves the widest scope of accessibility among all modifiers.

For example:

```
public class JavaBook
public Book() {.....}
```

default

If any modifier is not set, then it follows default accessibility. No specifier keyword is required for it. The **default** modifier is accessible only from within the package and not from outside it.

For example:

```
class JavaBook
Author() {.....}
```

protected

The **protected** access specifier is declared when you want to make the members of a class accessible in the class that defines them and also in other classes which inherit from that class.

For example:

```
protected class JavaBook
protected Author() {.....}
```

private

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

For example:

```
private class JavaBook
private Author() {.....}
```

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. It starts with two forward slashes (//).

For example:

```
//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. Multiple line comments are enclosed within /* and */. Anything between these marks is ignored by the compiler.

For example:

```
/*This is a multiple lines comment. Anything  
inside it is ignored by the compiler.*/
```

WRITING THE FIRST JAVA PROGRAM

There are many editors available in the market to write and run a Java program. You can either use an editor 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.

To write a Java program, first of all, open the Notepad editor from the **Start** menu and write the program in it and then save the program with the file name **FirstProgram.java**.

Example 1

The following example is a simple Java program. The program will print Hello Java on the screen.

```
// The program will print Hello Java  
1  class FirstProgram  
2  {  
3      public static void main(String args[ ])  
4      {  
5          System.out.println("Hello Java");  
6      }  
7  }
```

Explanation

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

Line 1

class FirstProgram

This line indicates the creation of a class with the name **FirstProgram**.

Line 2

{

This line indicates the start of the definition of the **FirstProgram** class.

Line 3

public static void main(String arg[])

This statement is used in every Java program and is the main method of Java. This is the entry point of every program. 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 allows the **main()** method to be called without creating a particular instance of the class. This is necessary because the **main()** method is called by the Java interpreter before any objects are created.

void

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

main(String args[])

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

Line 4

{

This line indicates the start of the definition of the **main()** method.

Line 5

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

This line will print Hello Java on the screen.

Line 6

}

This line indicates the end of the definition of the **main()** method.

Line 7

}

This line indicates the end of the definition of the **FirstProgram** class.



Note

The initial line in the program // The program will print Hello Java is not a part of program. It is a single line comment and for reference only.

Compiling and Running a Java Program

A Java program is easy to compile and run. You can compile a Java program in any command editor. Before compiling and running a Java 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. 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 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 class file name of the Java program which is generated after compilation of program. Note that while executing the **java** command, you do not need to add extension of the Java file.



Note

*It is a good practice to save your java program with the class name which is in the program to avoid errors. For Example, Class name in the program is **FirstProgram** then file name must be saved as **FirstProgram.java**.*

SETTING THE PATH OF PROGRAM DIRECTORY

To compile and run a program, it is necessary to set a path (location) to locate JDK binaries such as “javac” and “java”. The environment variable **path** is used to locate these JDK binaries.

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

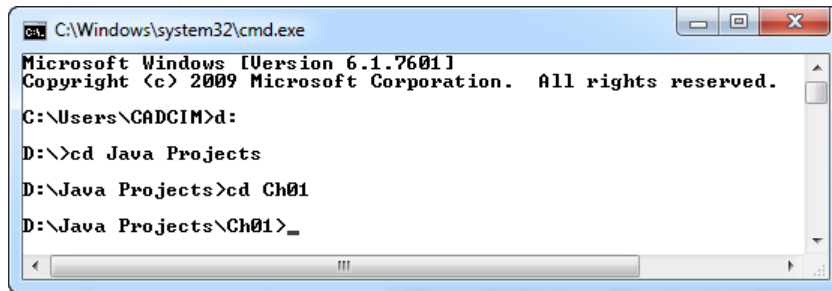
The following steps are required to set the path directory of your Java project on a temporary basis:

Step 1

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

Step 2

Open the command editor from the **Start** menu. Enter into the drive and directory, where you will save your file. For example, your root directory is *Ch01* which is inside *Java Projects* directory in the **D:** drive. You can do so by following the commands given in the Figure 1-9.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\CADGIM>d:
D:\>cd Java Projects
D:\Java Projects>cd Ch01
D:\Java Projects\Ch01>_
  
```

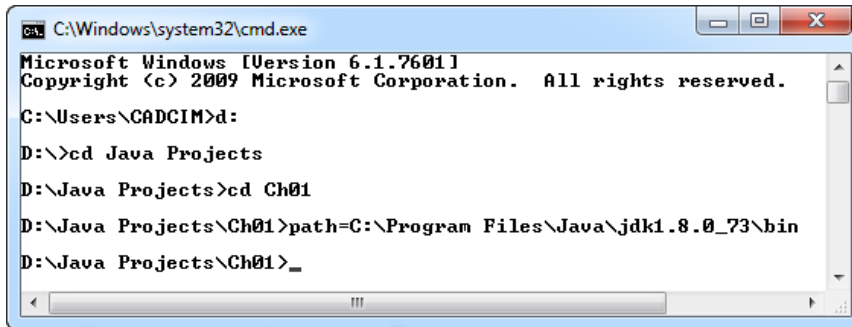
Figure 1-9 Commands to change the current drive and directory

Step 3

Write the following command at the command prompt:

```
path=The full path of bin directory of Java
```

Here, the **path** is a command. Write the full path of the bin directory of Java after the assignment operator and then press ENTER, as shown in Figure 1-10. The path will be set.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

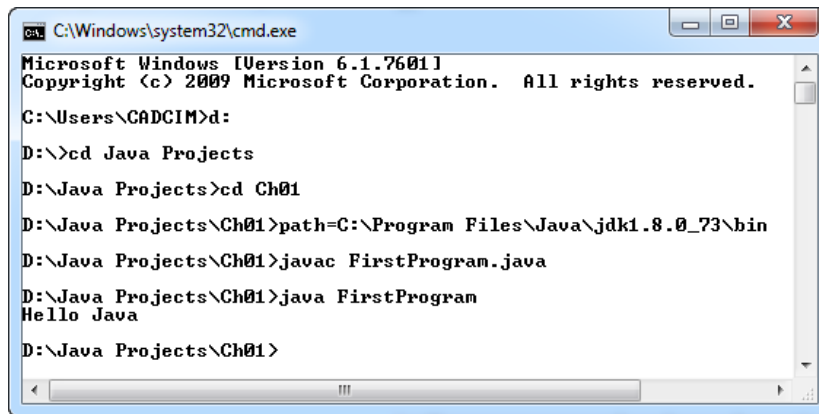
C:\Users\CADGIM>d:
D:\>cd Java Projects
D:\Java Projects>cd Ch01
D:\Java Projects\Ch01>path=C:\Program Files\Java\jdk1.8.0_73\bin
D:\Java Projects\Ch01>_
  
```

Figure 1-10 Setting the path of the bin directory on temporary basis

Step 4

Now, you can compile and run all your programs in your project directory. You can run Example 1 to test. Repeat the same process for compiling and running the program that you had learned earlier. The program will be executed without any errors.

The output of Example 1 after setting path on temporary basis is displayed in Figure 1-11.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\CADCIM>d:
D:\>cd Java Projects
D:\Java Projects>cd Ch01
D:\Java Projects\Ch01>path=C:\Program Files\Java\jdk1.8.0_73\bin
D:\Java Projects\Ch01>javac FirstProgram.java
D:\Java Projects\Ch01>java FirstProgram
Hello Java
D:\Java Projects\Ch01>
```

Figure 1-11 The output of Example 1 after setting path on temporary basis



Note

To go back to previous directory, you can directly write **cd** in command prompt.

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

Go to Control Panel from the Start menu. Next, click on the **System** icon; the **View basic information about your computer** window will be displayed, as shown in Figure 1-12. Alternatively, right-click on **Computer** from the **Start** menu and then choose **properties** to display the **View basic information about your computer** window.

Step 2

Next, choose **Advanced System Setting** option from the **View basic information about your computer** window; the **System Properties** dialog box will be displayed, as shown in Figure 1-13.

Step 3

In the **System Properties** dialog box, choose the **Advanced** tab and then choose the **Environment Variables** button; the **Environment Variables** dialog box will be displayed, as shown in Figure 1-14.

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.

Step 5

Enter **PATH** in the **Variable name** text box and enter complete path of the **bin** directory of your JDK in the **Variable value** text box and choose the **OK** button, as shown in Figure 1-15.

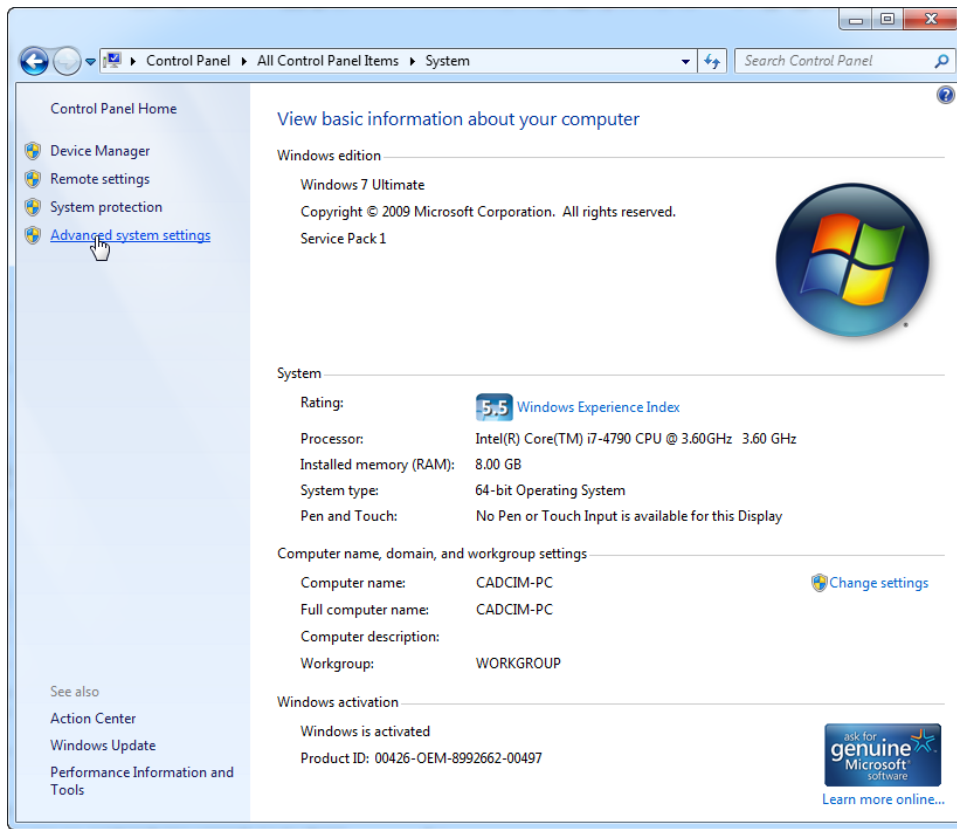


Figure 1-12 The *View basic information about your computer* window

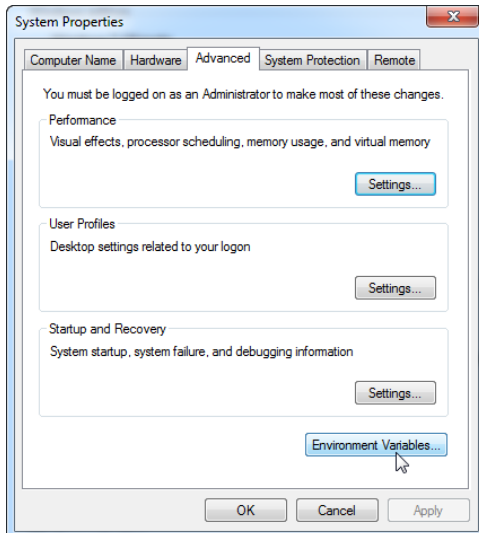


Figure 1-13 The *System Properties* dialog box

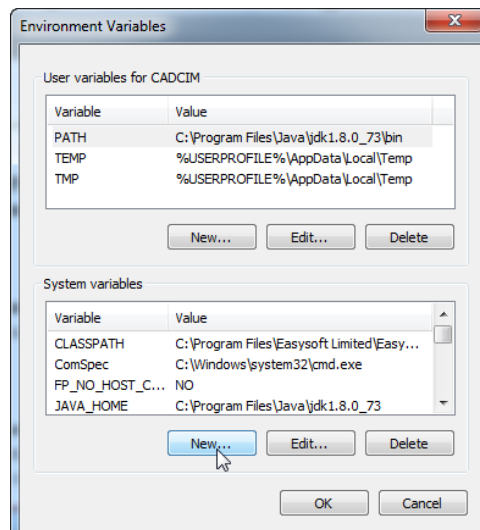


Figure 1-14 The *Environment variables* dialog box

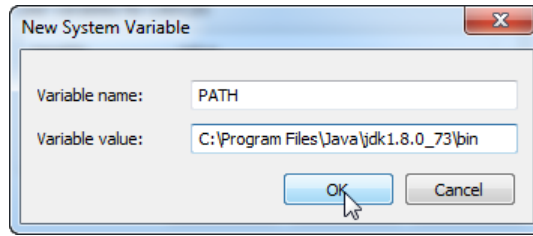


Figure 1-15 The New System Variable input box

Again, choose the **OK** button in the **Environment Variables** dialog box and then choose the **OK** button in the **System Properties** dialog 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.

The output of Example 1 after setting path on permanent basis is displayed in Figure 1-16.

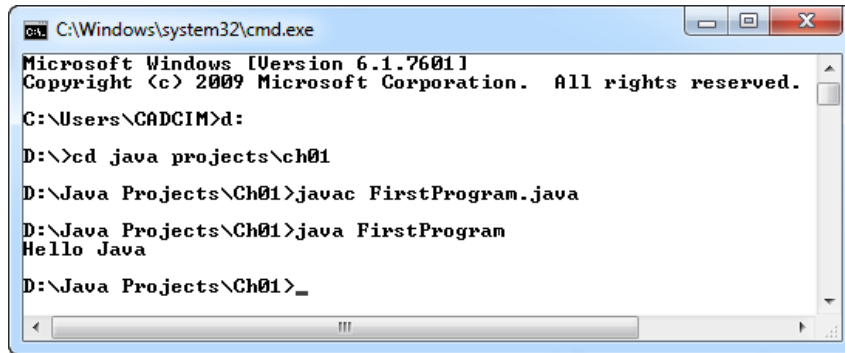


Figure 1-16 The output of Example 1 after setting path on permanent basis

INSTALLING NetBeans IDE

NetBeans is a popular free open source IDE for quickly and easily develop desktop, mobile, and web applications using Java, JavaScript, HTML5, PHP, C/C++, and much more. Most of the programmers use IDE because it eliminates the need for separate tools such as a text editor, a compiler, and a runner program.

An IDE integrates all these tools into a single toolset with a graphical user interface. There are many IDEs available on internet some of them have their own compilers and virtual machines.

To install NetBeans, navigate to <http://www.oracle.com/technetwork/java/javase/downloads/index.html> and then click on the **Download** button corresponding to the **NetBeans with JDK 8** entry, refer to Figure 1-17.

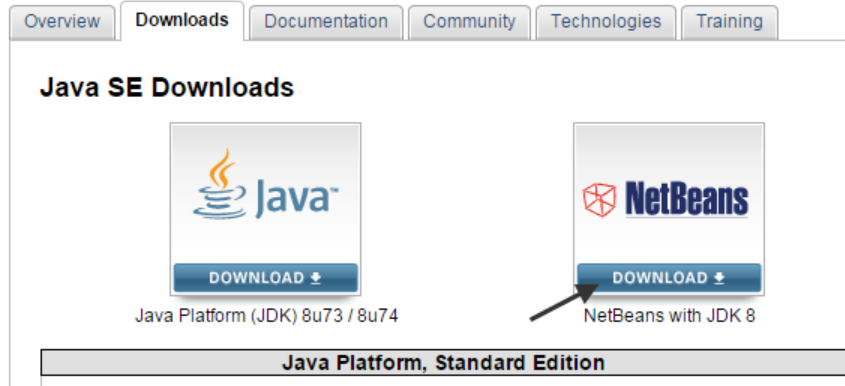


Figure 1-17 NetBeans download link

On the next page displayed, select the **Accept License Agreement** radio button and then click the **jdk-8u73-nb-8_1-windows-x64.exe** corresponding to the **Windows x64** entry. Save the file on your hard drive. If you have a 32-bit machine, click on the **jdk-8u73-nb-8_1-windows-i586.exe** link corresponding to the **Windows x86** entry. Double-click on the **jdk-8u73-nb-8_1-windows-x64** file; the installation process begins and you are prompted to accept the license agreement. Accept it and go ahead with the rest of installation process; the NetBeans shortcut icon will be placed on the desktop. Double-click on this icon; the NetBeans interface will be displayed, as shown in Figure 1-18.

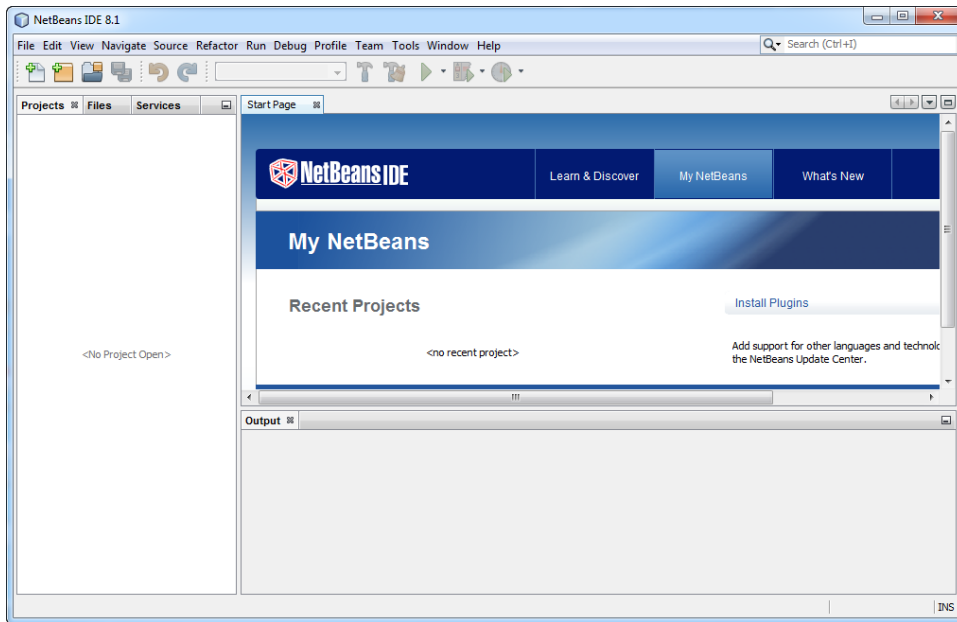


Figure 1-18 The NetBeans Interface

WRITING YOUR FIRST JAVA PROGRAM IN NETBEANS

The first step in writing the first Java program in netbeans is to create a new empty project in your preferred IDE, NetBeans in this case. To create a new project, choose **File > New Project** from the **NetBeans** menu bar to open the **New Project** dialog box, as shown in Figure 1-19.

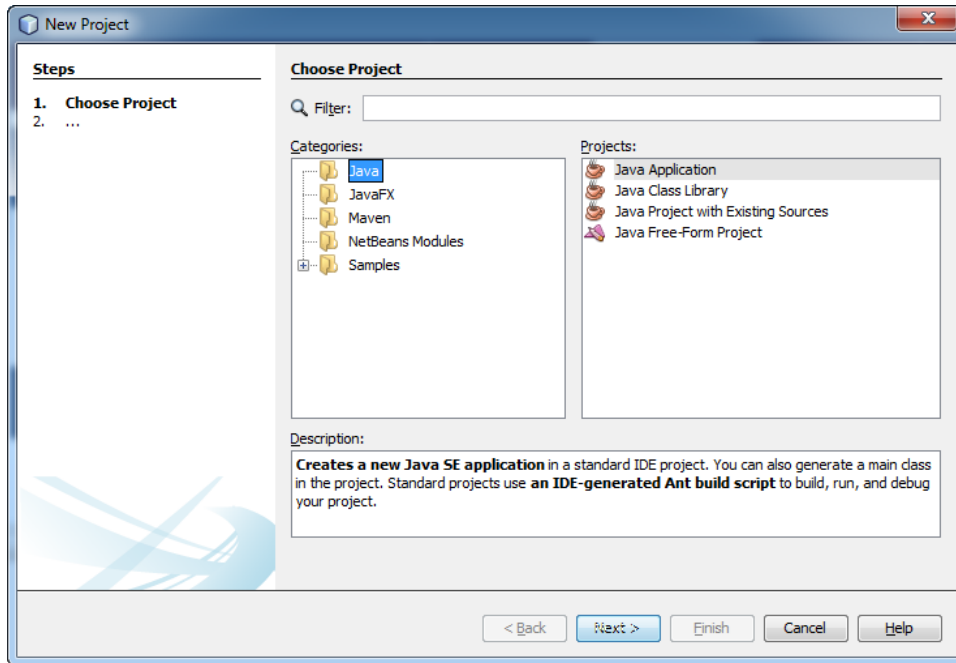


Figure 1-19 The New Project dialog box

In this dialog box, select **Java** in **categories** and **Java Application** from the **Project** list and then choose the **Next** button; the **New Java Application** dialog box will be displayed, as shown in Figure 1-20.

From the **Name and Location** area of the dialog box, enter the name of the project as **NetProgram** in the **Project Name** edit box. Choose the **Browse** button corresponding to the **Project Location** field if you want change the location of the projects and then choose the **Finish** button to close the dialog box and create the project.

The project is displayed in the Project window at the left in the interface, refer to Figure 1-21.

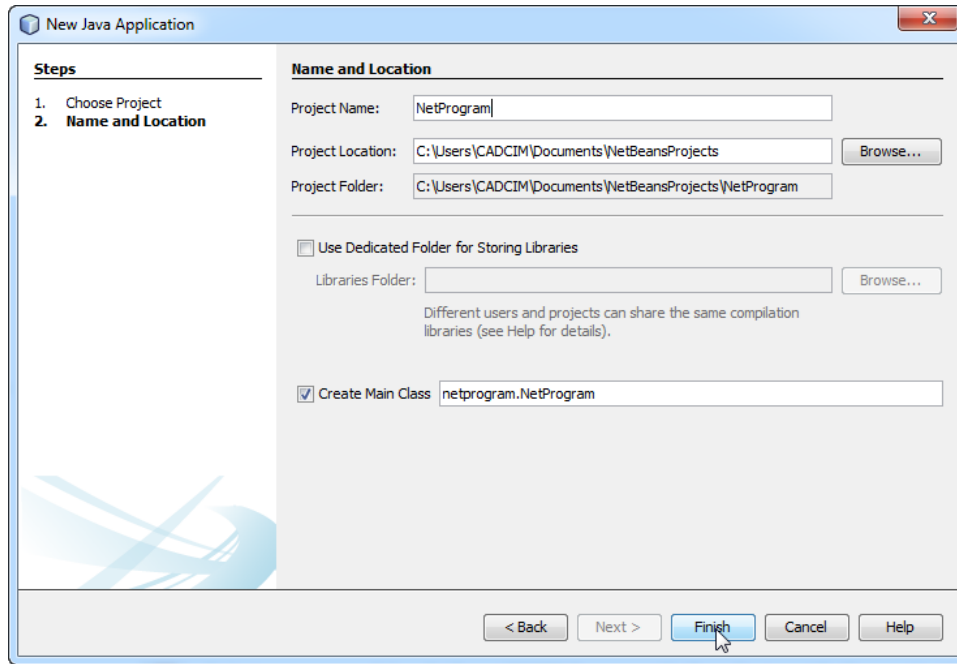


Figure 1-20 The New Java Application dialog box of Netbeans

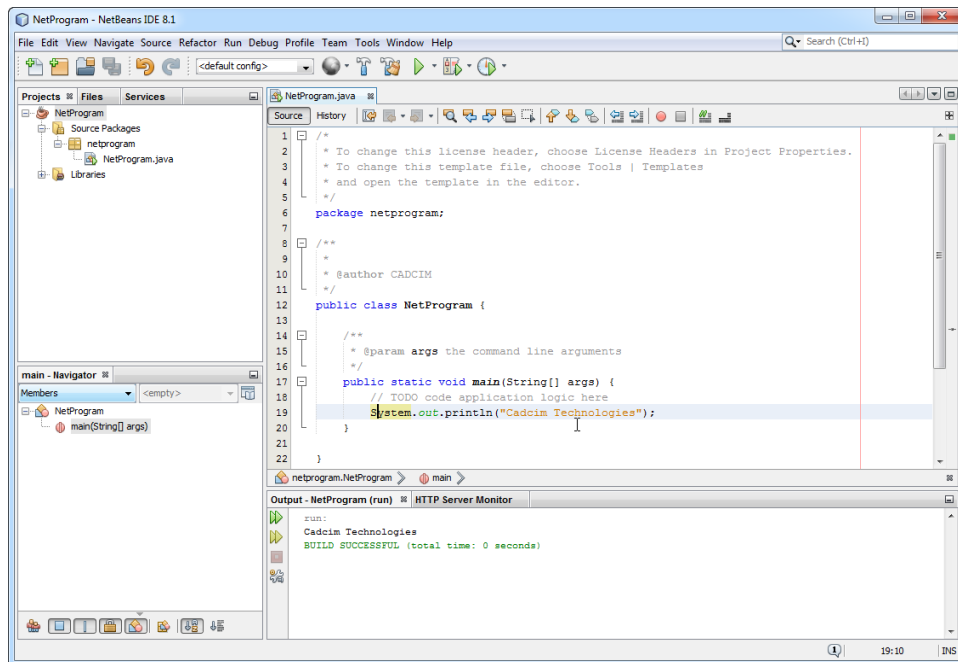


Figure 1-21 Project NetProgram displayed in the Project window

The following code is displayed in the **NetProgram.java** tab of the editor.

```
1  /*
2   * To change this license header, choose License Headers in Project
  Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package netprogram;

7   /**
8    *
9    * @author CADCIM
10   */
11  public class NetProgram {

12      /**
13       * @param args the command line arguments
14      */
15      public static void main(String[] args) {
16          // TODO code application logic here
17          System.out.println("Cadcim Technologies");
18      }
19  }
```

Explanation

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

Line 1 to 5

/*

*** To change this license header, choose License Headers in Project Properties.**

*** To change this template file, choose Tools | Templates**

*** and open the template in the editor.**

*/

These lines are multiline comment. Anything between /* and */ is ignored by the compiler.

Line 6

package netprogram;

This line defines the package of the class where **package** is the keyword and **netprogram** is the name of the package.

Line 7 to 10

/**

*

*** @author CADCIM**

*/

Again, these lines are multiline comment which is ignored by the compiler.

Line 11

```
public class NetProgram {
```

In this line, **public** is an access specifier which is accessible throughout the program. **class** is a keyword and **NetProgram** is the name of the class and the curly bracket indicates the start of the definition of the class.

Line 12 to 14

```
/**
```

```
* @param args the command line arguments
```

```
*/
```

Again, these lines are multiline comment which is ignored by the compiler.

Line 15

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

This line contains the **main()** method which is treated as the starting point of every Java program. The curly bracket indicates the start of the definition of the **main()** method.

Line 16

```
// TODO code application logic here
```

This is a single line comment which is ignored by the compiler.

Line 17

```
System.out.println("Cadcim Technologies");
```

This line delivers the output **Cadcim Technologies** on the screen when Java program runs.

Line 18

```
}
```

This line indicates the end of the definition of the **main()** method.

Line 19

```
}
```

This line indicates the end of the definition of the **NetProgram** class.

To run the project, choose **Run > Run Project** from the menu bar or press **F6**. On successful run, the **BUILD SUCCESSFUL** message will be displayed in the **Output** window at the bottom of the interface. If there are any error(s) in the program, a window will be displayed with errors. You need to fix the error(s) and then rebuild the program.

The following output is shown in the Output window at the bottom of the interface.

Output

Cadcim Technologies

BUILD SUCCESSFUL (total time: 0 seconds)

Self-Evaluation Test

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

1. What was the name initially given to Java ?
(a) Aok (b) Oak
(c) Ako (d) Oka
2. On which of the following platforms can Java run?
(a) Linux (b) Windows
(c) Mac (d) All of these
3. Which of the following inheritance is not supported by Java?
(a) Multilevel Inheritance (b) Multiple Inheritance
(c) Single Inheritance (d) Hierarchical Inheritance
4. Which of the following is supported by Java?
(a) Data Abstraction (b) Encapsulation
(c) Polymorphism (d) All of these
5. What do you call an inheritance, wherein multiple classes are derived from one super class?
(a) Single Inheritance (b) Multilevel Inheritance
(c) Hierarchical Inheritance (d) Multiple Inheritance
6. Which of the following commands is used to compile a Java program?
(a) **java** (b) **javap**
(c) **javac** (d) **jvm**
7. The term IDE stands for _____.
8. _____ helps in hiding the complexities of data.
9. In Encapsulation, methods and data are combined into a single unit. (T/F)
10. Java applets do not need any interpreter in order to execute. (T/F)
11. The **javac** command is used to interpret a Java program. (T/F)
12. While executing the **java** command, there is no need to add the extension file(.java). (T/F)

Review Questions

Answer the following questions:

1. What is byte code? How is it generated?
2. What is the use of **javac** and **java** commands?
3. What is JVM? How does JVM handle the byte-code?
4. What is the use of **import** keyword in java?
5. Check the syntax error in the following program and after correction, give the output of the program.

```
Class Hello
{
    Public static void main(String args[ ])
    {
        system.out.println("Hello")
        System.out.print('Welcome to the exciting world of Java');
    }
}
```

6. What is the difference between the following two statements?

```
System.out.print("Hello World!");
System.out.println("Hello World!");
```

EXERCISE

Exercise 1

Write a program to print the following statement on the screen:

Java is an interesting language.
It is easy to learn.

Answers to Self-Evaluation Test

1. b, 2. d, 3. b, 4. d, 5. c, 6. c, 7. Integrated Development Environment, 8. Data Abstraction,
9. T, 10. T, 11. F, 12. T