Overview of Developing Java Program

➢ Introduction

Java is a general purpose object oriented programming language. We can develop two types of java program:

1. Stand-alone applications programs
2. Web applets programs

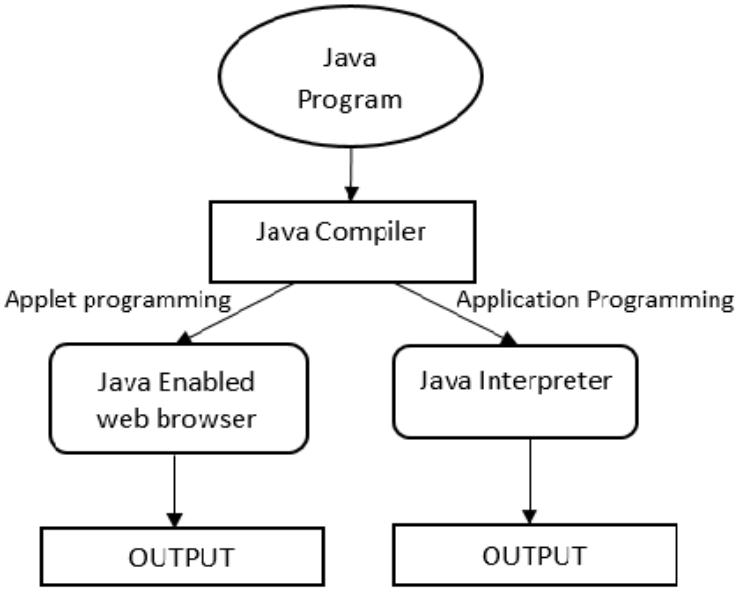


Fig2.1 Programming way of java

- **Stand-alone applications programs**

Stand-alone applications programs which is written in java-code that run as a separate computer process, not an add-on of an existing process. **Standalone program**, a **program** that does not require operating system's services to run. A portable **application**, which can **be** run without the need for installation procedure. All applications we can developed in java which were developed earlier in C and C++. For run this stand-alone java program involved two steps……

1.Compile Source code of java with javac Command (java compiler), it converts source code into byte code.

2. Execute this byte code by using java Command (java interpreter), it converts byte code into machine code.

- **Web applets programs**

Applets are small java program developed for the internet application. An applet is a program written in the **Java** programming language that can be included in an HTML page, and it is a small Internet-based program written in **Java**, a programming language for the Web, which can be downloaded by any computer. The **applet** is usually embedded in an HTML page on a Web site and can be executed from within a browser. It can be run on client side or server site. It requires web browser for running with two steps……

1.Compile Applet Source code of java with javac Command (java compiler), it converts source code into byte code.

2. Execute this byte code by using appletviewer Command (java web browser), it run the applet code

2.1 Structure of Java Program

The Stand-alone Java programming having one or more section of a structure of java program (shown in fig 2.2) it is depending on the application which we want to developed.It can be including one or more classes and interfaces, packages etc. An applet programming contains another structure which we will see further…

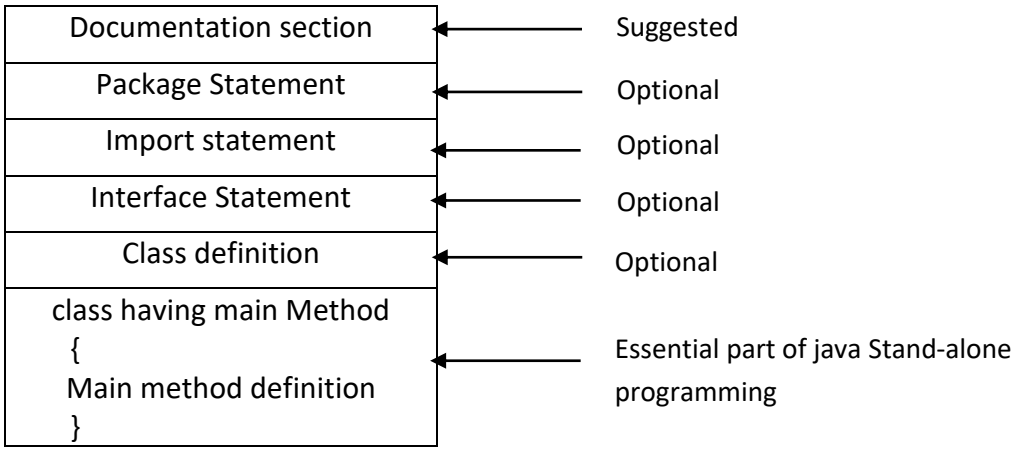


Fig 2.2: Structure of Java Program (Stand-alone Program)

- **Documentation section**

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

Sr.No.	Comment & Description
1	/* text */ :- The compiler ignores everything from /* to */.
2	//text:- The compiler ignores everything from // to the end of the line.
3	/** documentation */ This is a documentation comment and in general its called doc comment. The JDK javadoc tool uses doc comments when preparing automatically generated documentation

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

- **Package Statement**

This statement used to define the package name which tells the compiler that the classes are defined here to belonging to this package This statement is optional in the java program While creating a package, you should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

E.g.: - package students;

- **Import statement**

This statement is similar as #include statement in C Programming. The import statement in Java allows to refer to classes which are declared in other packages to be accessed without referring to the full package name.

Syntax import packagename.class;

Or

import packagename.*;

- **Interface Statement**

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. It is basically used for implementing the multiple inheritance.

- **Class definition**

Java program may contain multiple class definition. Classes are essential and primary element of java program. These classes are used to map the object with real world problem. You can define two or more classes in one program but only one class must be having main method

- **main method class**

This class is essential part of java Stand-alone programming. Every java program requires main method as a starting point of java program. A simple java program contains only this program. The main method creates an objects of various classes and establishes the communication between them. On reaching the end of main the program terminates and control passes back to operating system.

2.2 How to get input from User… (CLA, Buffered Reader File)

There are two ways to getting the input from user in java program.

1. By using Command Line Arguments (CLA)
2. By using Buffered Reader class.

➢ By using Command Line Arguments (CLA)

This is the 1st way to get the input from user, in the command line argument is the argument passed to a program at the time when you run it. To access the command-line argument inside a java program is quite easy, they are stored as string in String array passed to the args[] parameter of main() method. We can used agrs[] array of string type for storing the value which passed from command line (Dos). If parameters are string type the it can be directly stored in args[indexno.] but if

it is having another data type then we have to parse it (Covert one data type to another) by parsing methods

e.g. 1. String a=args[0]; // if value is string type then it stored in String type variable

2. int a=Integer.parseInt(args[0]);//if value is integer type then we have to convert it from String type to integer type by using parsing methods.

➢ By using Buffered Reader class.

- **Getting Simple User Input**

Thus far, the programs you have written have been one-sided in that they perform specific tasks and do not accept any user input with some massage. Every time you run these programs the output is exactly the same, making them all but useless in the eyes of a user after the first few times they are run. How can you make procedures more dynamic? By adding the functionality to accept and use user input. Anytime the user runs the application, he or she can enter different input, causing the program to have the capability to have different output each time it is run.

Because programmers write programs in the real world to be useful to users, this almost always means that the programs provide some interface that accepts user input. The program then processes that input and spits out the result. Accepting command-line input is a simple way to allow users to interact with your programs. In this section, you will learn how to accept and incorporate user input
into your Java programs.

- **Steps for using Buffered Reader Class:-**

1. Import IO package on top of the program
   By:- import java.io. *;
2. Create an object of BufferedReder Class
   Syntax:-BufferedReaderbr = new BufferedReader(new InputStreamReader(System.in));
3. Handled the error by using try- catch block //this will we see in unit 14
   Syntax:- try

```
{
Executable Statements which may occur Error …..
}
catch(Exception obj)
{
}
```

What follows is a listing of the HelloUser application:

```
/*
* HelloUser
* Demonstrates simple I/O
*/
import java.io.*;
public class HelloUser {
public static void main(String args[]) {
String name;
BufferedReader reader;
reader = new BufferedReader(new InputStreamReader(System.in));
System.out.print("\nWhat is your name? ");
try {
```

```
name = reader.readLine();
System.out.println("Hello, " + name + "!");
}
catch (IOExceptionioe) {
System.out.println("I/O Exception Occurred");
}
}
}
```

As you can see in Figure 2.6, the program prints a message asking for the user's name. After that the cursor blinks awaiting user input. In Figure 2.7 you see that the user entered her name, "Roseanne" and the application read it in and then printed "Hello, Roseanne!"

Using the BufferedReader Class

Unfortunately, getting simple user input is not very straightforward. One might think it would make sense that the syntax for reading a line would be as simple as writing a line of output. However, this is not the case. As you know, writing standard output can be done like this: System.out.println("Shampoo is better!"); So, it would be natural to think reading input could be done like this:

- System.in.readln();

But it is not that simple. First you must import the java.io package, which provides functionality for system input and output:

- import java.io.*;

Imp:- A package is a group of complimentary classes that work together to provide a larger scale of functionality. This is the second Java program you've written that uses the import statement. you imported the java.awt.Graphics class, which is part of the java.awt package. The difference here is that you use an asterisk to signify that you might be interested in all the classes that the java.io package groups together. Basically, you are importing the java.io package here to give your program the capability to call upon the I/O functionality. Specifically, it allows you to use the BufferedReader and Input- StreamReader classes in your program. At this point, you don't need to understand anything about these classes except that InputStreamReader reads the user's input and BufferedReader buffers the input to make it work more efficiently. You can think of a buffer as sort of a middle man.. When reading data in, a program has to make a system call, which can take a relatively long time (in computer-processing terms). To make up for this, the buffering trick is used. Instead of making a system call each time you need a piece of data, the buffer temporarily stores a chunk of data before you ask for it using only one system call. The buffer is then used to get data for subsequent requests because accessing memory is much faster than using a bunch of system calls.

- BufferedReader reader;

Neither of them is a primitive data type. They are both instances of classes. The name variable is declared to be a String, giving it the capability to hold strings, which are covered in the next section, and reader is declared to be a BufferedReader, giving it the functionality to buffer input. The next line basically specifies reader to be a standard input buffer: reader = new BufferedReader(new InputStreamReader(System.in)); After you have instantiated the BufferedReader object, which you have stored in the reader variable, it can be used to accept user input. This line of code is what prompts the user.
name = reader.readLine();

- Handling the Exceptions

Although exception handling is covered in more detail later in this book, I feel that I should explain the basics of it here because you are required to handle exceptions in this application. Exceptions are encountered when code does not work as it is expected to. Exception handling is used to plan a course of action in the event your code does not work as you expect it to. Here is an analogy to help you digest this concept. You have to go to work or school every normal weekday, right? Well, what if there is a blizzard or a tornado and you cannot go

to work? In this analogy, the blizzard or tornado is the exception because it is an abnormality. The way this exception is handled is that you end up staying home from work. Java requires that exceptions be handled in certain situations. In the HelloUser application, you are required to handle an IOException (input/output exception). What if you never actually get the user's input here? This would be an exception and you would not be able to incorporate the user's name in the program's output.

In the case of this application, you issue an error message, indicating that an exception was encountered. You do not say "Hello" to the user—you can't because you were not able to retrieve the user's name. You handled exceptions in the HelloUser application by using the try…catch structure. Any code, such as reading user input that might cause an exception, is placed within the try block, or "clause". Remember that a block is one or more statements enclosed within a set of curly braces:

```
try
{
name = reader.readLine();
System.out.println("Hello, " + name + "!");
}
```

Here, you are trying to read user input and use it in a standard output line. Whatif it doesn't work? That's what the catch clause is for: catch (IOExceptionioe)

```
{
System.out.println("I/O Exception Occurred");
}
```

If the code within the try clause does not work as it is expected to—there is an IOException— the code within the catch clause is executed. In this case the error message "I/O Exception Occurred" is printed to standard output to let the users know that a problem was encountered. In the real world, you try to handle exceptions as gracefully as you can. When detecting exceptions, you should try to handle them in a way so that your program uses default values instead of halting abruptly, but in some instances, your program just can't continue due to some error that you can't provide a work-around for. In these cases, you should at the very least, try to generate meaningful error messages so that users can use them to resolve any possible problems on their end.

- The Math Game In this section you incorporate much of what you've learned so far into a single application. After you write and compile this application, you can actually use it as a tool to remind yourself how arithmetic operators work in Java. Here is a listing of the source code for MathGame.java:

```
/*
* MathGame
* Demonstrates integer math using arithmetic operators
*/
import java.io.*;
public class MathGame {
public static void main(String args[]) {
int num1, num2;
BufferedReader reader;
reader = new BufferedReader(new InputStreamReader(System.in));
try {
System.out.print("\nFirst Number: ");
num1 = Integer.parseInt(reader.readLine());
System.out.print("Second Number: ");
num2 = Integer.parseInt(reader.readLine());
reader.close();
}
// Simple Exception Handling
catch (IOExceptionioe) {
System.out.println("I/O Error Occurred, using 1...");
```

```
num1 = num2 = 1;
}
catch (NumberFormatExceptionnfe) {
System.out.println("Number format incorrect, using 1...");
num1 = num2 = 1;
}
//Avoid this pitfall e.g. 1 + 1 = 11:
//System.out.println(num1 + " + " + num2 + " = " + num1 + num2);
System.out.println(num1 + " + " + num2);
```

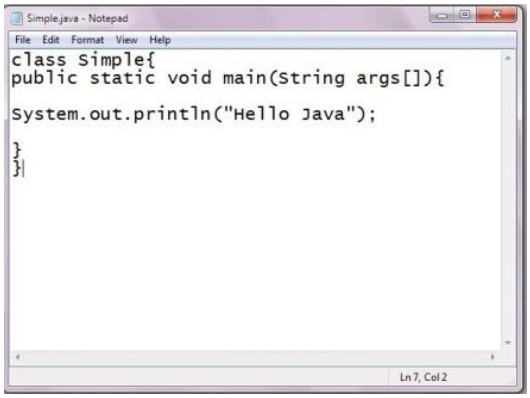
2.4 Simple first Java Program

➢ Syntax of java program

```
class class_name
        {
        public static void main(String args[])
                {
                Statement 1……;
                .
                Statement N……..;
                }
        }
```

Parameters used in first java program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't retrn any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument.



Writing Your First Application

The best way to learn Java is to jump right into it. In this section, you write astand-alone application that can run on any system that has a Java interpreter.You write the source code, learn how to compile it, and then run it. Keep in mind as you do this that you will be coming back to it and analyzing it afterward, sodon't worry if you don't get it right away. In fact, you probably won't understandit until I go over with you what you did.

Hello, World!

This application is as basic as it gets. It is the typical first program used in programmingbooks for many different languages. Basically, the HelloWorld applicationdemonstrates how to code a simple Java program by printing a message tothe screen. Take a look at the source code:

```
/* Hello World
* The classic first program
*/
public class HelloWorld
{
public static void main(String args[])
{
System.out.println("Hello, world!");
}
}
```

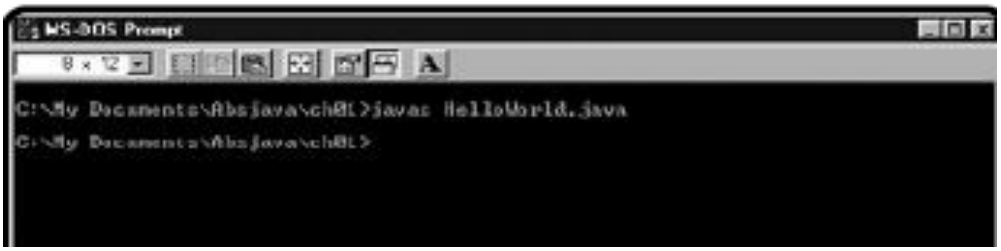
All that you need to do to write your first Java application is to copy this sourcecode into the text editor of your choice (I used Notepad). To do this, create a filenamed HelloWorld.java and type this source code in the file. This file, as well asall the other source code, is available on the CD-ROM; however, for your own benefit,I urge you to actually type the source code yourself. You get a better feel forthe language that way. After you finish typing this code, save the file.

Compiling the Program

[After you saved the HelloWorld source code, you need to compile it before it will run. The command instructing the JDK to compile your code is javac. First, open your command prompt window, such as Windows' MS-DOS prompt. Make sure you are in the directory (folder) that contains your HelloWorld.java source file. If you are unfamiliar with navigating your directory structure from a command prompt environment, here's a brief explanation of the cd command. You use the cd command to change directories in DOS. If you want to enter a subdirectory ofthe directory you are currently in, you type: cd *subdir* where*subdir* is the name of the subdirectory you want to enter. The command prompt by default indicates where you currently are in your directory structure. For instance, if you are in a directory named superdir that is on the C: drive and is not itself in any other directory, your command prompt looks like this: C:\superdir> And if you then typed cd subdir, your command prompt would look like this: C:\superdir\subdir> To back out of a directory, type cd.. at the command prompt. To get out of all directories and back to the root directory, type cd\. UNIX works similar to this except the slash is a forward slash (/) instead of a backslash (\).Next, at your operating system's command prompt (such as the DOS prompt),
type:]

```
javac HelloWorld.java
```

Make sure that you are in the proper directory and that you have the Java Developer'sKit (JDK) installed. Figure 1.3 demonstrates a successful compile, whereasFigure 1.4 demonstrates the importance of naming your file correctly. If you arehaving trouble with this step and you are sure that JDK is installed, make surethat you have copied the HelloWorld source code exactly and that your file isnamed **HelloWorld.java.**

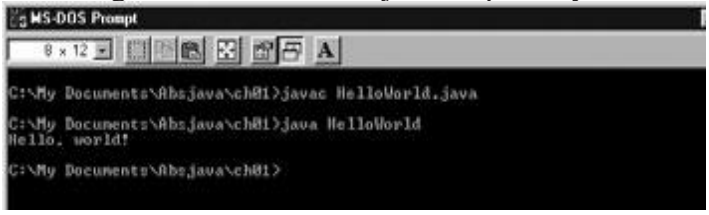


Running the Application

Now that you have written and compiled your program, how do you see the product of your efforts? To run the program from Windows, type the following at yourcommand prompt:**java HelloWorld**

If you're a Mac user, double-click on your new HelloWorld.class file. The JavaRunner application will then prompt you for any arguments (you don't need anyjust yet). It will then display the output, "Hello, world!" in a pop-up window titled*stdout*, short for standard output.

After you do that, your Java interpreter will kick into action and you should seesomething similar to what is shown in Figure 1.5. Congratulations! You've justcompleted your first Java program!



Learning Java Syntax Basics

The set of rules you must follow when writing program code are referred to asJava *syntax*. For now, after basics of Java syntax, it becomes easier to learn new features of theJava programming language because the syntax remains consistent and is thereforeintuitive. At the beginning of the code, notice the forward slash followed byan asterisk and some text. This is one way to add comments to your code.

```
/*
* Hello World
* The classic first program
*/
```

The next line of code is a blank line, which as you might imagine, does nothing.Following that is the beginning of your program class definition:

- **public class HelloWorld {**

It is must define a class in every Java program. Basically, a *class* is a group of functionsand characteristics in Java code that can be reused. **The public keyword**,used as a modifier that describes other classes' access to this class, is notrequired, but it is a good idea to use it anyway. This way you will start off withgood coding habits. It is not necessary at this point to fully understand the publickeyword. The same goes for the class keyword. It is used to identify HelloWorld as a class.The open brace denotes the beginning of the contents of your class definitionand the corresponding closing brace (the last line of the program) denotes theend of the class definition. To sum it up, this line of code starts a Java programcalled HelloWorld, and all the code within the braces makes up its guts.The next line of code starts the main() method definition.

- **public static void main(String args[]) {**

Methods are groups of statements that are executed by your computer when instructed to do so. For now, it is importantthat you know that the **main ()** method actually drives Java applications. It isthe first place the Java VM looks when you run your application. The group ofstatements within this method, collectively called a *block statement*, is defined within its own set of braces. Within the braces of this method, you define a listof steps you want the computer to do when running your program.The one statement inside the **main() method,**

Which is your next line of code,instructs your computer to say hello to the world.System.out.println("Hello, world!");This is actually a call to a method that handles your system's standard output. Itprints one line to the screen and adds a carriage return. You can refer to the JavaAPI documentation for classes System and PrintStream for more detailed information
about how standard output works. Inside the parentheses, within quotes,**Including Comments**adding comments to your code is not required, although it is definitely a goodpractice. Comments help you or anyone reading your code understands what yourprogram does. This can greatly facilitate the debugging process. You will be ableto read your comments later and remember what the different components ofyour procedure are. Comments also help when you go back and add new functionalityto your code because you will be less likely to be confused by what youhad previously done. There are two basic types of comments in Java—single-linecomments and multi-line comments. If you just want to make a note about a particularline of code, you usually precede that line of code with a single-line commentas shown here:

```
//The following line of code prints a message using standard output
System.out.println("Hello, World!");
```

Single-line comments start with double slashes //. This tells the compile to disregardthe following line of code. After the double slashes, you can type anythingyou want to on that single line and the compiler will ignore it.Sometimes you might want to write a comment that spans more than one lineof code. You can precede each line with double slashes if you choose to, but Javaallows you to accomplish this more easily. Simply start your comment with aslash followed by an asterisk: /*. You can type anything you want to after this,including carriage returns. To end this comment, all you need to do is type */.

```
/* I just started a comment
I can type whatever I want to now and the compiler will ignore it.
So let It be written
So let It be done
I'm sent here by the chosen one
so let It be written
so let It be done
to kill the first born pharaoh son
I'm creeping death
from Metallica's song, Creeping Death
I guess I'll end this comment now */
```

Everything in between the start and end of this comment is considered free text.This means you can type anything you want to within them. If you take anotherlook at the HelloWorld source code, you will notice that I used a multi-line comment.I typed the name of the HelloWorld program and then followed with severalmore lines. I preceded every line with an asterisk. You don't have to do this.I only did it to make the comments stand out more. Feel free to develop your ownstyle of commenting your code, but keep in mind that you or someone else

- **The *main()* Method**

When running a Java application, the main() method is the first thing the interpreterlooks to. It acts as a starting point for your program and continues to drive
it until it completes. Every Java application requires a main() method or it willnot run. In this section, I point out what you should understand about it becauseyou'll be using it often. In fact, you use it in every application you write.The main() method always looks just like it did when you programmed theHelloWorld application.

- **public static void main(string args[]) {**

Let's take a closer look at the parts of the **main()** method:
• The **public** and static keywords are used in object-oriented programming.Simplified, public makes this method accessible from other classesand static ensures that there is only one reference to this method usedby every instance of this program (class). Static methods are also referredto as class methods, because they refer to the class and not specificinstances of the class. This is a difficult concept to understand at thispoint. You can simply gloss over it for now.
• The **void** keyword means that this method does not return any valuewhen it is completed.
• main is the name of the method, it accepts a parameter—String args[].Specifically, it is an array (list) of command-line arguments. As is the casewith all methods, the parameters must always appear within the parenthesesthat follow the method name.
• Within the curly braces of the **main ()** method, you list all the operationsyou want your application to perform. I stated earlier that every Java applicationrequires a main () method; however, it is possible to write a Javasource code file without defining a main () method. It will not be consideredan application, though, and it won't run if you use the java commandon it. Now your head might be spinning at this point. Don't let this entire make you forget what I stated earlier. The main () method is simplythe driver for your application.

2.2 How to get input from User… (CLA, Buffered Reader File)

2.3 Output Statement in Java

2.4 First Java Program

2.5 Classes, Objects and Methods

2.6 How to Save and Execute Java program