

Introduction to Java

Java programming language is a high-level, object-oriented, general-purpose, and secure programming language. It was developed by James Gosling at Sun Microsystems in 1991. At that time, they called it OAK.

Sun Microsystem changed the name to Java in 1995. In 2009, Oracle Corporation took over Sun Microsystem.

Java is the most widely used programming language. It is designed for the distributed environment of the Internet. Java is freely accessible to users, and we can run it on all the platforms. Java follows the WORA (Write Once, Run Anywhere) principle, and is platform-independent.

Editions of Java

There are three editions of Java. Each Java edition has different capabilities. The editions of Java are:

1. Java Standard Editions (SE): We use this edition to create programs for a desktop computer.
2. Java Enterprise Edition (EE): We use this edition to create large programs that run on the server and to manage heavy traffic and complex transactions.
3. Java Micro Edition (ME): We use this edition to develop applications for small devices such as set-top boxes, phones, and appliances, etc.

Features of Java

1. Simple: Java is simple because its syntax is simple and easy to understand. Java eliminates many complex and ambiguous concepts of C++. For example, the use of explicit pointers and operator overloading are not in Java.
2. Object-Oriented: Everything in Java is in the form of the object. In other words, it has some data and behavior. A Java program must have at least one class and object.
3. Robust: Java always tries to check errors at runtime and compile time. Java uses a garbage collector to provide a strong memory management system. Features like Exception handling and garbage collection make Java robust or strong.

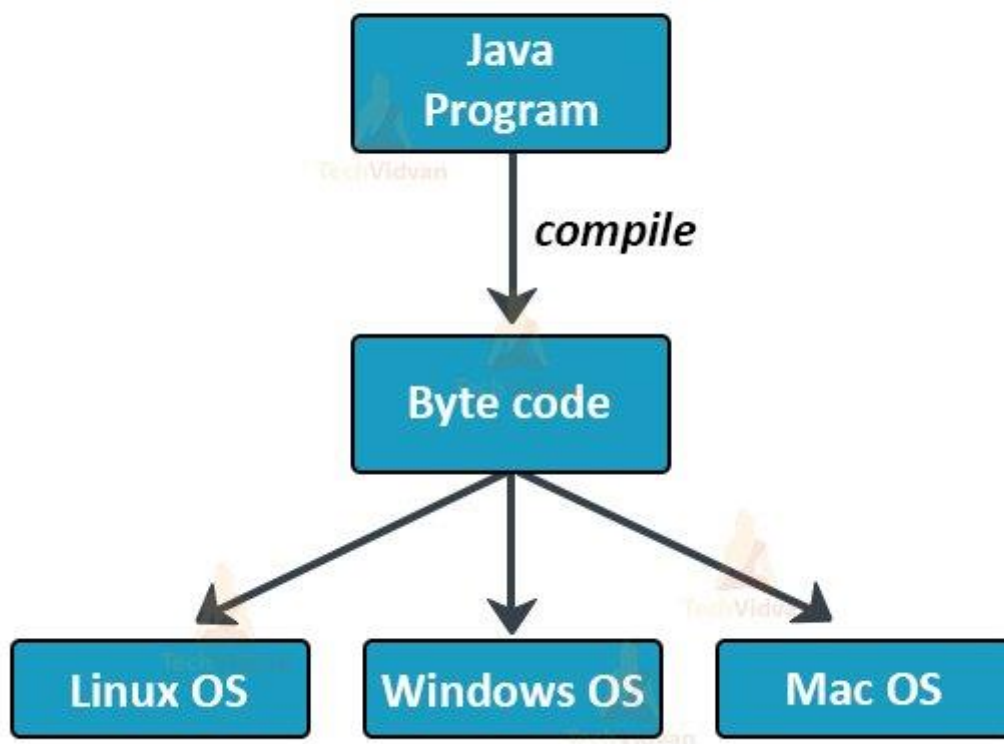
4. Secure: Java is a secure language, as Java does not use explicit pointers. All Java programs run in the virtual machine. Moreover, Java contains a security manager that defines the access levels of Java classes.

5. Platform-Independent: Java provides a guarantee that to write code once and run it anywhere(at any platform). The compiled byte code is platform-independent, and we can run it on any machine irrespective of the Operating system.

6. multithreaded – supports multi-threaded programming for writing program that perform concurrent computations

7. interpreted and high-performance – Java programs are compiled into an intermediate representation – bytecode:

Java Platform Independent



6. Portable: We can carry the bytecode of Java to any platform. There are no implementation-dependent features in Java. Java provides predefined information for everything related to storage, such as the size of primitive data types.

7. High Performance: Java provides high performance with the use of the Just-In-Time (JIT) compiler.

8. Distributed: Java is a distributed language as it provides networking facilities. Java works very well in the distributed environment of the Internet. This is because Java supports TCP/IP protocol.

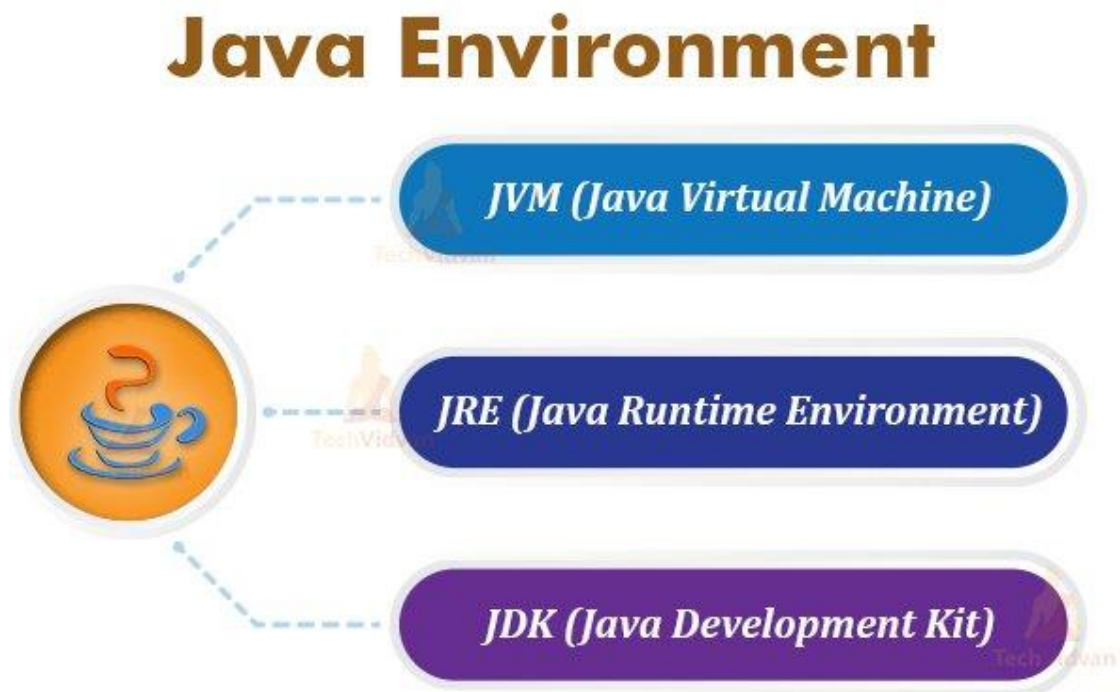
EJB(Enterprise Java Beans) and RMI(Remote Method Invocation) of Java are used to create a distributed system.

9. Multi-threaded: Java is a multi-threaded language as it can handle more than one job at a time.
Applications of Java Programming

Java is a widespread language. The following are some application areas in which we find Java usable:

1. Desktop applications
2. Web applications
3. Mobile applications (Android)
4. Cloud computing
5. Enterprise applications
6. Scientific applications
7. Operating Systems
8. Embedded systems
9. Cryptography
10. Smart cards
11. Computer games
12. Web servers and application servers

Java Environment- JVM, JRE, and JDK



1. JVM (Java Virtual Machine)

Java Virtual Machine provides a runtime environment in which we can execute the bytecode. JVM is platform-dependent. It performs the following tasks:

- Loading the code
- Verifying the code
- Executing the code
- Providing a runtime environment

2. JRE (Java Runtime Environment)

JRE is a collection of tools. These tools together allow the development of applications and provide a runtime environment. JVM is a part of JRE. JRE is also platform-dependent like JVM.

3. JDK (Java Development Kit)

Java Development Kit provides an environment that helps to develop and execute the Java program. There are Development Tools in JDK to provide an environment to develop Java programs.

JDK, along with the JRE, contains other resources like the interpreter, loader, compiler, an archiver (jar), and a documentation generator (Javadoc). These components together help you to build Java programs.

Object-Oriented Concepts in Java

As we all know that Java is an object-oriented language, there are many concepts of the same. Some object-oriented concepts in Java are:

1. Abstraction

An abstraction is a technique of hiding irrelevant details from the user, and showing only the necessary ones. For example, the driver knows how to drive a car; he does not need to know how the car runs. In Java, we can achieve abstraction using abstract class and interface.

2. Encapsulation

Encapsulation is the process of wrapping up data and functions/methods into a single unit. An example of encapsulation is the class in Java that contains both properties and methods.

3. Inheritance

Inheritance is the mechanism by which one class acquires all the features of and properties from another class. We can achieve Inheritance in Java by using the 'extends' keyword. Inheritance facilitates the reusability of the code.

4. Polymorphism

Polymorphism is the capability to occur the same thing in multiple forms. In other words, Polymorphism states single action in different ways. For example, a girl in the classroom behaves like a student, in house behaves like a daughter.

There are two types of polymorphism in Java: Runtime polymorphism(Method Overriding) and Compile-time polymorphism (Method Overloading).

Java Keywords

Keywords are the special words that are basically reserved keywords in any programming language. We cannot use them in the rest of the programs. We can only use them as the name of variables in Java, class, or method. Java has around 50 keywords which are basics to java:

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

Java Variables

Computer programs read data from input devices like keyboard, mouse etc. They process this input data and write it to an output device or network. Java stores the program data in variables.

Java program first declares the variables, reads data into these variables, executes operations on the variables, and then writes them somewhere again.

There are the following types of variables in Java basics:

1. Local Variables
2. Class Variables (Static Variables)
3. Instance Variables (Non-static Variables)

Java Data Types

There is a basic java data type for each variable in Java. The data type of a variable determines the type of data the variable can contain, and what operations we can execute on it.

Every bit of processed data every day is divided into types. The type of data is called a data type. There are various kinds of data types in Java.

Broadly, the data types are mainly of two categories:

a. Primitive Data Types in Java

Primitive data types are fundamental data types offered by Java. These are the basic data values. These data types are hard coded into the Java compiler so that it can recognize them during the execution of the program.

There are 8 types of primitive data types in Java:

- a. int
- b. float
- c. char
- d. boolean
- e. byte
- f. short
- g. long
- h. double

b. Non-Primitive Data Types in Java

Non-Primitive data types are the reference data types. These are the special data types that are user-defined. The program already contains their definition. Some examples of non-primitive or reference data types are classes, interfaces, String, arrays, etc.

Java Operators

Java Operators are the special type of tokens. When they are coupled with entities such as variables or constants, they result in a specific operation. The operation can be any, such as addition, multiplication or even shifting of bits, etc.

There are the following types of Java operators;

- Arithmetic Operators
- Logical Operators
- Unary Operators
- Assignment Operators
- Ternary Operators
- Relational Operators
- Bitwise Operators
- Shift Operators
- instance Of operator

Java Method

A method or function basically defines a behavior. There can be a number of methods in Java. In methods, there are logics written. We can manipulate data in methods and also execute actions on them.

They have a syntax similar to classes:

```
< returnType > <methodName >
{
    action1;
    action2;
}
```

For Example:

```
void print()
{
System.out.println("Hey I am learning Java at MES");
}
```

Comments in Java

Comments are needed whenever the developer needs to add documentation about a function that is defined within the program. This is to enhance code readability and understandability. Comments are not executed by the compiler and simply ignored during execution.

The comments are of the following types:

a. Single-Line Comments in Java

The single-line comments consist of a single line of comment. We generally write them after a code line to explain its meaning. We mark the single-line comments with two backslashes(//).

For Example:

```
class SingleLineComment //Declaring the class
{
    public static void main(String[] args)
    {
        String str = "MES College"
        //Declaring string with the value-"MES College"
    }
}
```

b. Multi-Line Comments in Java

Multi-line comments, as the name suggests, span for multiple lines throughout the code. We generally write to them at the beginning of the program to elaborate on the program.

Developers also use them to comment out blocks of code during debugging. We mark them using starting tag(/*) and an ending tag(*/).

For Example:

```
class MultiLineComment
{
    public static void main(String[] args)
    {
        /* All this is under a multiline comment
```



```

        The compiler won't execute it
        Thank you */
    }
}

```

Java Class

Class in Java is the blueprint that defines similar types of objects derived from it. A class represents a set of methods and properties that are common to all the objects.

A class is one of the fundamental building blocks of Object-Oriented programming in Java. We can define a class in Java using the class keyword.

The syntax of the class is:

```

< access - specifier > class < ClassName >
{
    instance variables;
    class method1() {}
    class method2() {}
} //end class

```

Example:

```

public class Main {
    int number;
    void test() {
        System.out.println("MSc Cs 2022 admission");
    }
}

```

Java Object

An object in java is an identifiable entity that has some characteristics and behavior. We create objects from class in Java.

For example, a Fan is an object that has three characteristics: It has three blades, It has a brown color, etc. Its behavior is: it rotates at some speed. We can create the object of a class once we define a class.

The syntax of creating objects in Java is:

```

< className > <objectName > =new < className> ();

```

Example:

```

Main Java = new Main ();

```

Object Creation in Java

Object in Java can be created in any of the following ways using:

- new operator
- new instance

Access Modifiers in Java

Access modifiers in Java enable users to limit the access of the entities or data they are defined with. Java provides the following access specifiers:

1. public: There is no restriction of access and the data is accessible to every class or interface inside the same or different packages.
2. private: The private specifier allows entities to be accessible only inside the class in which we declare them.
3. protected: The class members declared with the protected keyword are accessible to classes within the same package or subclasses of different packages.
4. default: If there is no access modifier mentioned then Java uses the default access modifier. This access specifier limits the access within the same package only.

Loops in Java

Loops in Java



Loops are the iterative statements that run a particular set of programs for a fixed number of times. Some of the types of iterative statements in Java are:

i. For loop in java

The 'for' loop causes a code snippet to run for a predetermined number of times. In a single statement, there is initialization, updation and test condition expressions.

For Example:

```
for (int num = 0; num < 5; num++)  
{  
    System.out.println("Hello");  
}
```

This prints Hello five times on the output screen

ii. Java While loop

The while loop runs indefinitely until the condition becomes false.

For Example:

```
while (num < 6)  
{  
    System.out.println("Hello");  
    num++;  
}
```

This prints Hello on the screen five times until the value of num becomes 6

iii. Java do-while loop

The do-while loop works the same as the while loop. The only is that in the do-while loop, the execution occurs at least once even if the condition is false.

For Example:

```
do  
{  
    System.out.println("Hello");  
}  
while ( num > 6 );
```

Conditional Statements in Java

Conditional Statements in Java



Conditional statements are statements that are purely based on the condition flow of the program. There are mainly three types of conditional statements in Java:

i. Java if statement

The if statement suggests that if a particular statement results in true then the block enclosed within the if statement gets executed.

For example:

```
if (condition)
{
//action to be performed of the condition is true
}
```

ii. Java if-else statement

The if-else statement states that if a particular condition is true then the if block gets executed. If the condition is false, then the else block gets executed.

Example:

```
if (condition) {
//action1, if conditions true
```

```
}  
else {  
    //action2, if condition is false  
}
```

iii. **Java Else if statement (else if ladder)**

The else if statement encloses an if statement within an else blocks.

Example:

```
if (condition)  
{  
    action 1  
}  
else if (condition2)  
{  
    action 2  
}
```

iv. **Java Switch case**

The switch case is responsible for checking multiple conditions. The switch case is based on the value of the variable that we pass in the switch statement. The value of the variable shows the flow of the control to either of the case blocks that we write.

Example:

```
switch (variableName)  
case value1:  
    action1;  
    break;  
case value2:  
    action2;  
    break;  
default:  
    action3;  
    break;
```

String in Java

Java String is a reference data type that represents text enclosed within double quotes(" "). String is a sequence of characters, that are useful to print any text or message. Strings are immutable, i.e. we cannot change them once created. The string always ends with a null character or '\0'.

There are two ways to create a Java string are:

- Using a string literal in Java
String name = "MES College Marampally";
- Using a new keyword
String name = new String("MES College Marampally");

Jump Statements in Java

Jump Statements are the statements that help us to continue or discontinue a loop during the execution of a program. There are three types of jump statements in Java:

i. Break statement in Java

The 'break' statement or the 'break' keyword breaks the nearest loop inside which it is present. The execution of the program continues from the next line just when the current scope ends.

ii. Continue Statement in Java

The continue keyword or the statement continues the execution of the program from the next iteration of the loop. It causes the loop to skip the current iteration.

iii. Return statement in Java

The return statement is generally useful in Java methods. When the function completes its execution, it returns the value. After the execution of the return statement, the remaining statements of the function after the return statement does not execute.

/* This is a simple Java program.

FileName : "HelloWorld.java". */

```
public class HelloWorld
{
    //Your program begins by calling the main().
    //Prints "Hello, World" to the terminal window.
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

The above program consists of three primary components: the class definition, main() method and comments. The following section will give you a basic understanding of this code:

1. Class definition: We use the 'class' keyword to declare a new class. We can also use the access specifier like public before the class keyword:

```
public class HelloWorld
```

2. Hello World is the name of the class that is an identifier in Java. The class definition contains the members of the class that are enclosed within the curly braces{ }.

3. Java Main() method: Every application in Java programming language must contain a main() method whose signature is:

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

public: We declare the main method as public so that JVM can execute it from anywhere.

static: We declare the main method as static so that JVM can call it directly without creating the object of the class.

Note: We can write the modifiers public and static in any order.

void: The main method does not return anything, therefore we declare it as void.

main(): main() is the name that is already configured in the JVM.

String[]: The main() method accepts a single argument which is an array of elements of type String.

The main method is the entry point for any Java application as in C/C++. The main() method will subsequently invoke all the other methods required by the program.

Below is the next line of code. It is present inside the main() method:

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

This line actually prints the string "Hello World" on the screen, followed by a new line on the screen. We can get the output on the screen because of the built-in println() method.

The System is a predefined class in Java. This class provides access to the system. out is the variable of type output stream that is connected to the console.

Important Points

The name of the class in the program is HelloWorld. This name is the same as the name of the file, HelloWorld.java. These same names are not a coincidence. In Java, the whole code must reside inside a class. And there must be at most one public class that contains the main() method.

By convention, the name of the class containing the main method should match the name of the file that holds the program.

Compiling Java Program

1. Firstly, we need to set up the environment. After that, we can open a terminal or command prompt in both Windows or Unix and can go to the directory/folder where we have saved the file: HelloWorld.java.
2. Now, to compile Java HelloWorld program, we need to execute the compiler: javac, specifying the name of the source file on the command line, like:
3. The compiler creates the compiled file called HelloWorld.class in the present working directory. This class file that contains the bytecode version of the program.

Running Java Program

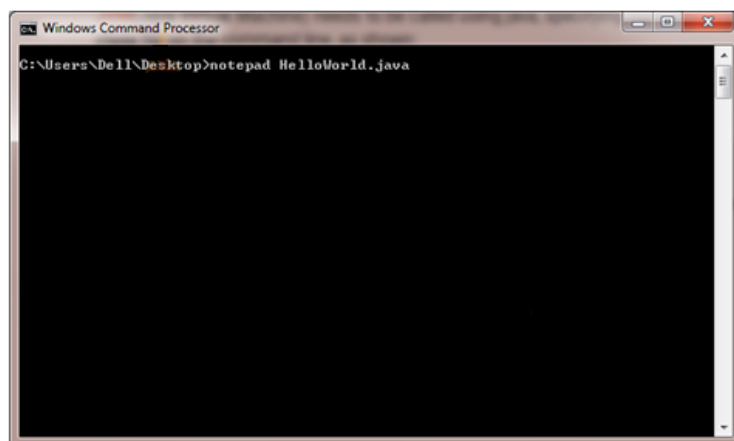
To execute Java program, we need to call JVM(Java Virtual Machine) using java command. After that, we specify the name of the class file on the command line, like:

```
java HelloWorld
```

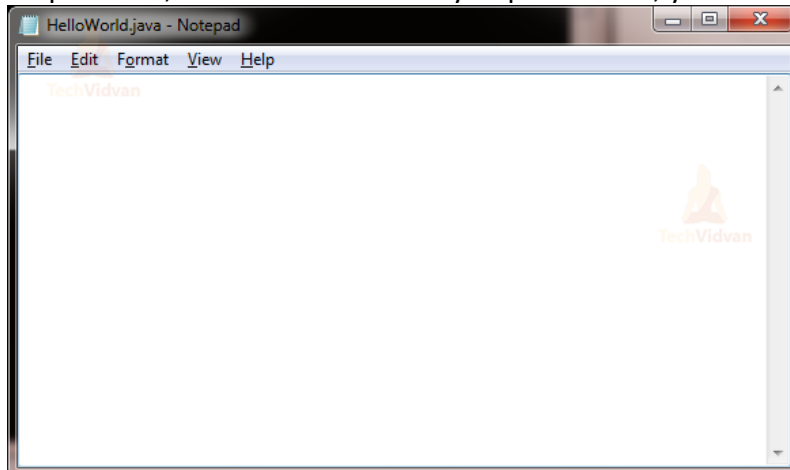
This will print "Hello World" on the terminal screen.

Steps to write HelloWorld Program in Windows

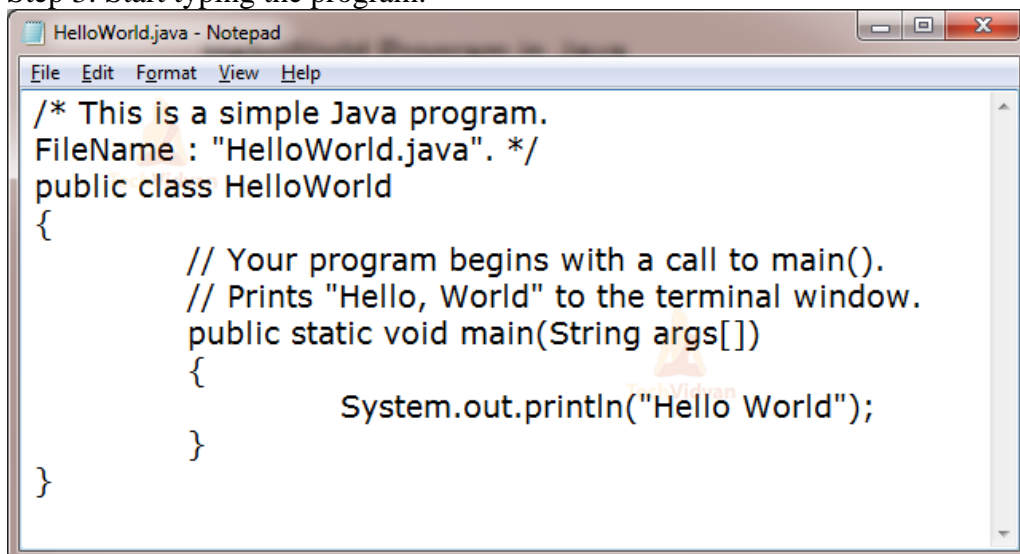
Step 1: Open Command Prompt Window, Reach to the desired folder and type notepad HelloWorld.java, like this:



Step 2: Now, hit Enter. As soon as you press enter, you will see a notepad editor screen:

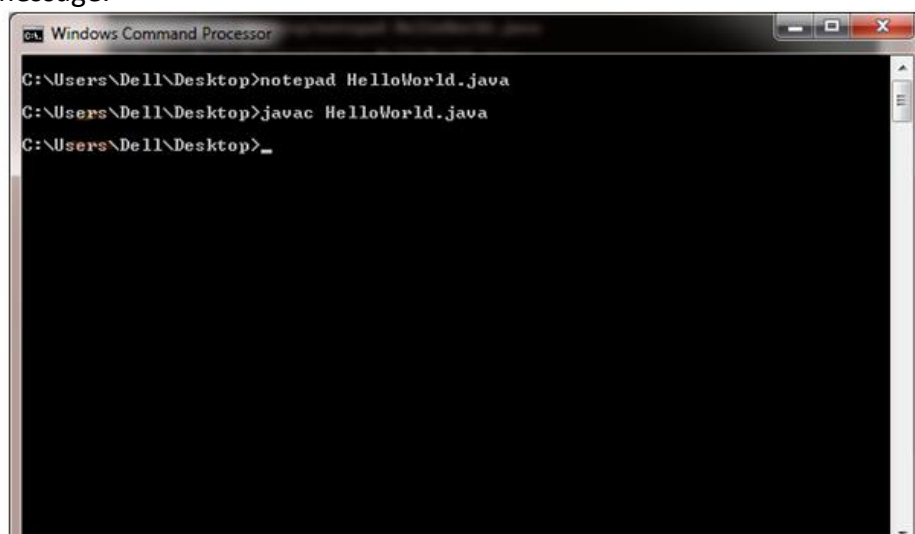


Step 3: Start typing the program:

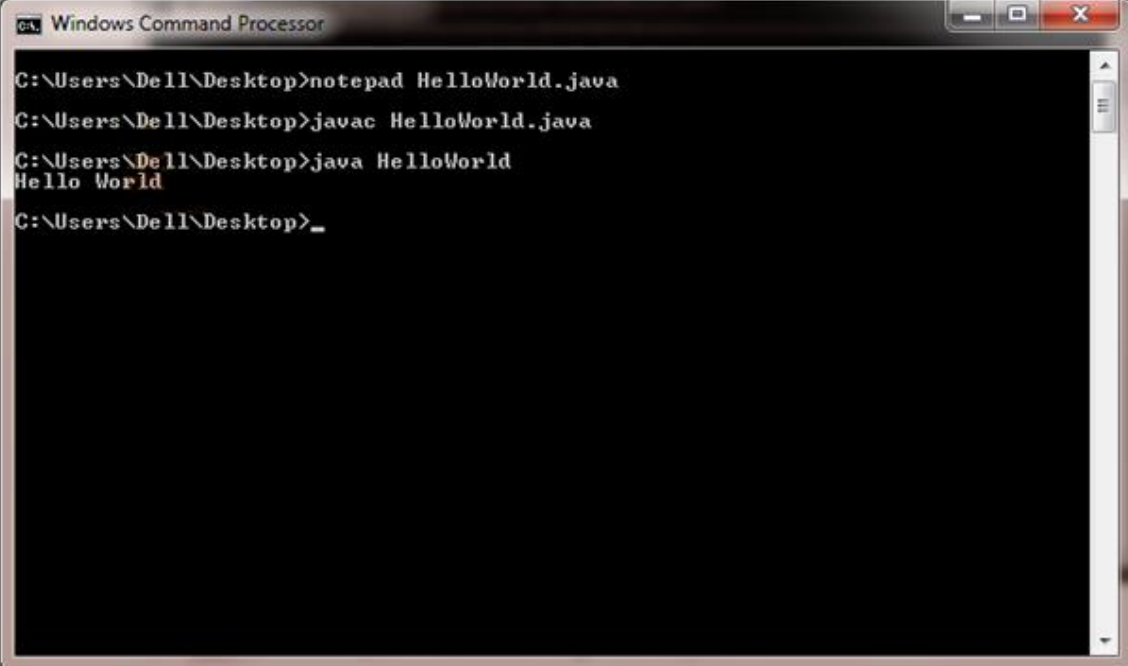


Step 4: Go to the File option and click the Save button to save the file. Close the notepad window. Move to the command prompt window again.

Step 5: Type here `javac HelloWorld.java`, and enter. If the program compiles successfully, then the cursor will start blinking on the next line, otherwise, there will be an error message.



Step 6: Now type java HelloWorld. Press Enter to get the output:

A screenshot of a Windows Command Processor window. The title bar reads "Windows Command Processor". The command prompt shows the following sequence of commands and output:

```
C:\Users\Dell\Desktop>notepad HelloWorld.java
C:\Users\Dell\Desktop>javac HelloWorld.java
C:\Users\Dell\Desktop>java HelloWorld
Hello World
C:\Users\Dell\Desktop>_
```

The output "Hello World" is displayed on a separate line after the command "java HelloWorld". The cursor is at the end of the last command line.

```
C:\Users\Dell\Desktop>_
```

You can see the printed output as Hello World on the screen.