Chapter 1: An Introduction to Java

What Is Java?

- Java is an object-oriented programming language developed by Sun Microsystems in 1991 at BELL Laboratory.
- Initial name of JAVA was OAK.
- James gosling who is one of the inventor of java language.
- Java has both compiler as well as interpreter.
- Java language was developed for internet purpose.
- Java language is more popular because it supports to the three

Key Elements :

- Applet
- Powerful programming language constructs
- Rich set of significant object classes.

Features of Java

1. **<u>Java Is Platform-Independent</u>**

   Java is completely platform independent programming language. Platform independence means the capability of the same program to work on different operating systems. That means we can execute our java program in windows to Linux O.S. and vice versa.

2. **<u>Java Is Portable</u>**

   Java is Portable means that programs written in Java can be compiled once and
   run on any machine.Platform-independence refers to a program’s capability to run on various *com*puter systems without
   the necessity of being recompiled.

3. Java Is Object-Oriented

Java is a real object-oriented language, which enables you to create flexible, modular programs. It includes a set of class libraries that provide basic data types, system input andoutputcapabilities, and other utility functions.

It also provides networking, common Internet protocol, image handling, and user-interface toolkit functions.

Java’s object classes provide a rich set of functionality, and they can be extended and modified to create new classes specific to your programming needs.

4. Java Is Easy to Learn

Java language is very easy to learn for the professional programmer or to the students those having knowledge of C++.

It is also very simple because it left out many unnecessary features such as pointer, implicit type casting, Structures or unions, operator overloading, templates, header files and multiple inheritance

5.Java Is Safe

Java provides security on several different levels.
First, the language was designed to make it extremely difficult to execute damaging code. The elimination of pointers is a big step in this regard. By eliminating all pointers except for a limited form of references to objects, Java is a much more secure language.

Another level of security is the bytecode verifier. Before a Java program is run, a verifier checks each byte code to make sure that nothing suspicious is going on.

In addition to these measures, Java has several safeguards that apply to applets. To prevent a program from committing random acts of violence against a user's disk drives, an applet cannot, by default, open, read, or write files on the user's system.

6. **Strictly Typed**

All objects and variables used in a program must be declared before they are used. This enables the Java compiler to locate and report type conflicts.

7. **Compiled and Interpreted**

Before you can run a program written in the Java language, the program must be compiled by the Java compiler.

The compilation results is a byte- code file that is similar to a machine-code file, can be executed under any operating system that has a Java interpreter.

This interpreter reads in the Byte-code file and translates the byte-code commands into machine- language commands that can be directly executed by the machine that's running the Java program. Therefore Java is both a compiled and interpreted language

Note :- Byte Code is not machine code. Byte code is get generated after the compilation of program and machine code is get generated after interpretation of the program.

8.Multi-threaded

Thread is a part of program. i.e. smallest unit of program is called Thread.

Thread is small unit of program which is in executive mode.

Java programs can contain multiple threads of execution, which enables programs to handle several tasks concurrently.

For example, a multi-threaded program can render an image on the screen in one thread while continuing to accept keyboard input from the user in the main thread. All applications have at least one thread, which represents the program's main path of execution.

9.Garbage collected

Java programs do their own garbage collection, which means that programs are not required to delete objects that they allocate in memory. This relieves programmers of virtually all memory-management problems.Garbage collection is a process it automatically deletes all the objects that are not used by any java program for long time.

10. Robust

It is related with memory management. In c++, memory is allocated to the variable by using new operator and de-allocated by using delete operator. In c memory is allocated to the variable by using malloc() function and deallocated by using free() function. In java memory is allocated to the variable or object by using new operator but memory deallocation is done automatically because of ROBUST.

Because the Java interpreter checks all system access performed within a program and java does not support direct memory access therefore Java programs cannot crash the system. Instead, when a serious error is discovered, Java programs create an exception. This exception can be captured and managed by the program without any risk of bringing down the system.

11.Extensible

Java programs support native methods, which are functions written in another language, usually

C++. Support for native methods enables programmers to write functions that may execute faster than the equivalent functions written in Java. Native methods are dynamically linked to the Java program; that is, they are associated with the program at runtime. As the Java language is further refined for speed, native methods will probably be unnecessary.

12.Distributed

Java facilitates the building of distributed applications by a collection of classes for use in networked applications.

By using Java's URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

13. Dynamic

Because it is interpreted, Java is an extremely dynamic language. At runtime, the Java environment can extend itself by linking in classes that may be located on remote servers on a network (for example, the Internet).

This is a tremendous advantage over a language like C++ that links classes in prior to runtime.

Comparison of Java and C++

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given on next slide :

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

1)JDK (Java Development Kit) Environment and Tools:

The Java Development Kit contains a variety of tools and Java development Environment. Following is a list of the main components of the JDK :

- Runtime interpreter
- Compiler
- Applet viewer
- Debugger
- Class file disassembler
- Header and stub file generator
- Documentation generator

The Compiler

The Java compiler is a command-line utility. The Java compiler (javac) is used to compile Java source code files into executable Java bytecode classes. In Java, source code files have the extension .java.

The Java compiler takes files with this extension and generates executable class files with the .class extension. The compiler creates one class file for each class defined in a source file.

<u>Syntax for the Java compiler follows</u> :

javac Options Filename

The Filename argument specifies the name of the source code file you want to compile. The Options argument specifies options related to how the compiler creates the executable Java classes

The Runtime Interpreter

The Java runtime interpreter (java) is used to run standalone Java executable programs in compiled, bytecode format. The runtime interpreter acts as a command-line tool for running Java programs that are either nongraphical or that manage their own window frame (applications); graphical programs requiring the display support of a Web browser (applets) are executed entirely within a browser.

The syntax for using the runtime interpreter follows :

java Options ClassName Arguments

The ClassName argument specifies the name of the class you want to execute. If the class resides in a package, you must fully qualify the name. For example, if you want to run a class called Roids that is located in a package called ActionGames, you execute it in the interpreter like this:

```
java ActionGames.Roids
```

The Arguments argument to the interpreter specifies the arguments passed into the main() method. For example, if you have a Java class called TextFilter that performs some kind of filtering on a text file, you would likely pass the name of the file as an argument, like this:

```
java TextFilter SomeFile.txt
```

The Options argument specifies options related to how the runtime interpreter executes the Java program.

The Applet Viewer

The applet viewer is a useful tool for testing Java applets in a simple environment. The applet viewer is a tool that serves as a minimal test bed for final release Java applets. You can use the applet viewer to test your programs instead of using a full-blown Web browser. You invoke the applet viewer from a command line like this:

appletviewer Options URL

The URL argument specifies a document URL containing an HTML page with an embedded Java applet. The Options argument specifies how to run the Java applet. There is only one option supported by the applet viewer: -debug. The -debug option starts the applet viewer in the Java debugger, which enables you to debug the applet.

Example :

appletviewer example1.html

example1.html is the HTML file containing the embedded Java applet.

The Debugger

The Java debugger (jdb) is a command-line utility that enables you to debug Java applications. The Java debugger uses the Java Debugger API to provide debugging support within the Java runtime interpreter. The syntax for using the Java debugger follows:

jdb Options

The Options argument is used to specify different settings within a debugging session.

The Header and Stub File Generator

The Java header and stub file generator (javah) is used to generate C header and source files for implementing Java methods in C. The files generated can be used to access member variables of an object from C code. The header and stub file generator accomplishes this by generating a C structure whose layout matches that of the corresponding Java class. The syntax for using the header and stub file generator follows : **javah** Options ClassName

The ClassName argument is the name of the class from which to generate C source files. The Options argument specifies how the source files are to be generated.

The Class File Disassembler

The Java class file disassemble (javap) is used to disassemble executable Java class files. Its default output consists of the public data and methods for a class. The class file disassemble is useful in cases where you don't have the source code for a class, but you want to know a little more about how it is implemented. The syntax for the disassemble follows:

javap Options ClassNames

The ClassNames argument specifies the names of one or more classes to be Disassembled. The Options argument specifies how the classes are to be disassembled.

The Documentation Generator (javadoc)

The Java documentation generator (javadoc) is a useful tool for generating API documentation directly from Java source code. The documentation generator parses through Java source files and generates HTML pages based on the declarations and comments. The syntax for using the documentation generator follows:

javadoc Options Filename

The FileName argument specifies either a package or a Java source code file. In the case of a package, the documentation generator creates documentation for all the classes contained in the package. The Options argument enables you to change the default behavior of javadoc.

Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

Types of Variables

There are three types of variables in Java :
1.local variable
2.instance variable
3.static variable

1. Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2. Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3. <u>Static variable</u>

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Operators in Java

Operator in Java is a symbol that is used to perform operations. For example: +,-, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

Decision Making statements

1. if statements
2. switch statement

Loop statements

1. do while loop
2. while loop
3. for loop
4. for-each loop

Jump statements
1.break statement
2.continue statement

Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.