Introduction To Java

1.1. History of Java

Java began life as programming language **Oak**

Java was developed at **Sun Microsystem**. The history of Java can be traced back to a project called Green project in SUN. **Patric Naughton, Bill Joy, James Gosling, Mike Sheridan** were some of the people working on the Green project.

The aim of this project was to develop software for smart consumer electronic devices such a complex remote controls. At first developer tried using C++ to develop the software but they faced a number of problems.

So James Gosling developed a new language, which he called Oak. A study showed that there already was a programming language called **Oak.** So they changed the name to **Java.**

In 1994, the Internet and the World Wide Web became popular. The people at Sun realized that Java was suitable to develop applications for the Internet.

They wanted fundamentally a new way of computing, based on the power of **networks and wanted the same software to run on different computers,** consumer gadgets and other devices

During that time **Mosaic** the **First Graphical Browser** was released, with this people on different types of machines and operating systems started accessing the applications available on the web.

Patrick Naughton and his colleague Jonathan Payne developed a browser in Java called **WebRunne**r, which they later renamed as **Hot Java**. Sun Formally released Java and Hot Java in 1995. Java proved to the world that it was possible to develop complex applications.

Members of the Oak team realized that java would provide the required **cross-platform independence** from the hardware network, and the operating system. Very soon, **Java became an integral part on the web.**

1.2 The New Era of Java

Java is a new programming language from Sun MicroSystems with elements from **C, C++,** and other languages, and with libraries highly tuned for the **Internet Environment.**

Java is quickly recognizable to many programmers. The statements and expressions are similar to those in many languages and in most cases identical to those in C and C++.

1.3 Comparison of Java with C & C++

Although Java adds some new things, it is most distinctive for what is left out.

1.3.1 Compared with C

Java does not have

- Pointer Manipulation
- Goto statement
- Automatic type conversion
- Global functions variables
- Type definitions aliases (typedefs)
- Preprocessor

1.3.2 Compared with C++

Java does not have

- Operator overloading

1.4 Characteristics of Java

Java is a High-Level Programming Language that is all of the following:

- Simple and Powerful
- Object-Oriented
- Architecture-Neutral
- Portable
- High-performance
- Interpreted
- Robust
- Multithreaded
- Distributed
- Dynamic
- Security

Java is....

Simple and Powerful

Java was designed to be easy for the professional programmer to learn and use effectively. Java makes itself by not having surprising features.

Since it exposes the inner working of a machine, the programmer can perform his desired actions without fear. Unlike other programming systems that provide dozens of complicated ways to perform a simple task,

Java provides a small number of clear ways to achieve a given task.

Anyone can master Java with a little bit of programming experience. If the user already understands the basic concepts of object-oriented programming, learning Java will be much easier.

Object Oriented

Java was not designed to be source-code compatible with any other language. Java team gave a clean, usable, realistic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

Architecture-Neutral

The Java designers worked hard in attaining their goal **"write once run anywhere, anytime, forever"** and as a result the **Java Virtual Machine was Developed.**

Since Java is Architecture-neutral it generates bytecode that resembles machine code, and are not specific to any processor(CPU).

Portable

In Java, the same mechanism that gives security also helps in portability. Many types of computers and operating systems are in use throughout the world and are connected to the Internet. For downloading programs through different platforms connected to the Internet, some portable, executable code is needed.

Java's answer to these problems is its well-designed architecture.

Interpreted and High performance

Java enables the creation of cross-platform programs by compiling the code into an intermediate representation called Java byte code.

This code can be interpreted on any system that has a Java Virtual Machine. Most of the earlier cross-platform solutions are run at the expense of performance. Other interpreted systems, such as BASIC, Tel and PERL, suffer from almost insoluble performance deficits.

Java byte code was carefully designed so that it can be easily translated into active machine code for very high performance by using a **Just-In-Time Compiler (JIT)**.

Robust

Most program fail today because of two reasons:

- **Memory management**
- **Exceptional conditions**.

Thus, the ability to create robust programs was given a high priority in the design of Java. Java forces the user to find mistakes in the early stages of program development. At the same time, Java frees the user from having to worry about many of the most common causes of programming errors.

Java checks code at compilation time.

However, it also checks the code at run time.

Java virtually rectifies the problem of memory management by managing memory allocation and **automatic memory deallocation by providing garbage collection for unused objects.**

Java also handles **exceptional conditions** by providing object-oriented exception handling. In a well-written Java program, all run-time errors can- and should be managed by the program itself.

Multithread

Java was designed to meet the real-world requirements of creating interactive, networked programs. To achieve this,

Java supports multithread programming, **which allows the user to write programs that perform many functions simultaneously.**

The Java run-time system enables the user to construct smoothly running interactive systems.

Distributed

Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. This allowed objects on two different computers to execute procedures(programs) remotely.

Java has recently reweived these interfaces in a package called Remote Method Invocation (RMI).

Dynamic

Java program carry with them extensive amounts of run-time information that is used to verify and resolve accesses to objects at run time. Using this concept it is possible to dynamically link code. Dynamic property of java adds strength to the applet environment, in which small fragments of byte code may dynamically updated on a running system.

Secure

Today everyone is worried about safety and security. People feel that conducting commerce over the Internet is as safe as printing their credit card number on the first page of a newspaper. Threatening of viruses and system hackers also exists. To overcome all these fears **Java has safety and security as its key design principle.**

Using Java Compatible Browser, anyone can safely download Java applets without the fear of viral infection or malicious intent.

Java achieves this protection by confining a Java program to the Java execution environment and by making it inaccessible to other parts of the computer.

We can download applets with confidence that no harm will be done and no security will be breached.

Java enables the constructions of virus-free, tamper-free systems.

1.6 The Java Application Programming Interface (API)

The Java API supports all kinds of programs with packages of software components that provide a wide range of functionality.

The core API given us the following features.

The Essentials: Objects, Strings, Threads, Numbers, Input and Output, Data Structures, System Properties, Data and Time, and so on.

Applets: The set of conventions used by Java applets.

Networking: URLs, TCP and UDP sockets, and IP addresses.

Internationalization: Help for the writing programs that can be localized for users worldwide. Program can automatically adapt to specific locales and be displayed in the appropriate language.

Security: Both low-level and high-level, including Electronic Signatures, Public/Private key management, Access control, and Certificates.

Software component: Known as Java Beans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Liver Connect.

Object Serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI)

Java Database Connectivity (JDBC): Provides uniform access to a wide range of relational databases.

Java not only has a core API, but also standard extensions. The standard extensions define APIs for **3D, Servers, Collaboration, Telephone, Speech, Animation,** and more.

Java API Packages

Packages	Purpose
`java.lang`	Core language classes
`java.applet`	Applet classes
`java.awt`	Graphics, window, and GUI classes
`java.awt.data transfer`	Data transfer (Eg. Cut & paste) classes
`java.awt.event`	Event processing classes & interfaces
`java.beans`	Java Beans component model API
`java.io`	Various types of input/output classes
`java.net`	Networking classes
`java.rmi`	Remote method invocation classes
`java.security`	Security classes
`java.sql`	JDBC SQL API for database access
`java.util`	Various useful data classes

1.7 Java Development Kit(JDK)

JDK is a collection of tools that are necessary to work with Java Applet and Applications.

- **javac -** the complier, checks for syntactical error and converts syntax free java program into bytecode.
- **java -** the Interpreter, executes the bytecode
- **appletviewer -** used to test applets on machines which does not have java enabled web browser

1.8 Type of Java Programs ?

Java is of two things

- **Application Programming language**
- **Applet**

1.8.1 Applications

Java applications can read from and write to files on the local computers. Standalone applications make use of the resources of just a single computer. It is stored on the hard disk of the local computers.

1.8.2 Applets

Applets are Java programs that are used to develop web pages.
Java applets are programs that require a Web Browser with a JVM to run.

Summary

JAVA is a fully object oriented programming Language developed by SUN MicroSystems in1995.

Advantages of JAVA

1. It is easy to learn.
2. Everything in JAVA is in the form of **Classes and Objects.**
3. It provides the features of **Abstraction, Encapsulation, Polymorphism and Inheritance.**
4. JAVA programs can access data across a network.
5. The byte code can be executed on a variety of computers running on different operating system (Platform independent).
6. Java does not allow a programmers to manipulate the memory of the system.
7. Java programs are faster when compared to programs written in other interpreter-based languages.
8. It allows multiple parts of a program to run simultaneously.
9. There are two types of JAVA programs **1.Application 2.Applets**

2. Java Programming & Fundamentals

2.1 Structure of JAVA program

```
//documentation section
import packages;
class <class_name>
{

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

Program

```
/* this is a simple java program .
      call this program as FirstProgram.java */

import java.lang.*;
class FirstProgram
{
      //program begins with a call to the main method

      public static void main(String args[])
      {
      System.out.println("Welcome to JAVA");
      }
}
```

2.2 Comments

A comments describes or explains the operation of the program to anyone who is reading the source code.

The compiler ignores the contents of the comment.

In java, **Three Types** of commment statements can be included in a programs.

They are:

(i) A **Single Comment** statement précised by all the text which follows the **//,** which treated as a comment and ignored by the compiler.

(ii) A **Multiline Comment** enclosed within the two character sequence **/*…*/**

(iii) The third type called the **Documentation Comment** begins with a **/** and ends with a */.** They are used to produce a HTML file that documents the program.

Every program in **Java** is a class. So, the first line of any Java program is always class followed by the name of the program. The name of the program can be anything it should only begin with an alphabet and contains no space.

The next line contains four words **`public`**, **`static`**, **`void`** and **`main`.**

`public`	The method `main` can be accessed by any other class.
`static`	The main method is declared as `static`, which means that the main method can be called without creating an object.
`void`	The method **`main`** does not return any value.
`main`	The program begins the execution at the main method
`String args[]`	This indicates that the command line arguments are passed into the method in an array of string called args.

Creating, Compiling and Running a JAVA Program

To create a Java program, open a text editor **Example:** Notepad

1. Type the Java program.
2. Ensure that the Java program is saved with the **same name** as given in the **class name** of the program. **Example:** **`FirstProgram.java`**
3. Also ensure that the program is saved with the extension **`.java`**
4. To compile the java program, type **`javac`** specifying the name of the source file on the command line.

   Syntax **`javac <filename>.java`**

   Example **`javac FirstProgram.java`**
5. The javac compiler creates a class file that contains the byte code version of the program.

   Example **`FirstProgram.class`**
6. To run the compiled java program, specifying the name of the class file on the command line which interprets the class file.

   Syntax **`java <class_filename>`**

   Example **`java FirstProgram`**

   Output **`Welcome to JAVA`**

2.3 Java Fundamentals

2.3.1 Java Keywords

There are 48 reserved keywords currently in the java language. These keywords, combined with the syntax of the operators and separators, form the definition of the java language.

abstract	do	implements	short
boolean	double	import	static
break	else	int	super
byte	extends	interface	switch
case	final	long	this
catch	finally	new	throw

`char`	`float`	`private`	`try`
`class`	`for`	`protected`	`true`
`continue`	`false`	`public`	`void`
`default`	`if`	`return`	`while`

Note These keywords **cannot** be used as variables, contants, classes or methods name.

2.3.2 Data Types

All variables or constants in the java language must have a data type.

A variable or constant data type determines the values that the variable can contain and the operations that can be performed on it.

Example Integers can contain only integral values (both positive and negative). You can perform operations, such as addition, subtraction on integer variables.

The java programming language has **two major categories of data types**

- **Primitive (Primary Data types)**
- **Reference (object)**

A variable of **primitive type** contains a single value of the appropriate size and format for its type: a number, a character, or boolean value.

Example the value of an **int** is an integer, the value of a **char** is a 16-bit Unicode character, and so on.

Arrays, classes, and Interface are Reference types. The value of a reference type variable, in contrast to that of a primitive type, is a reference to the actual value or set of values represented by the variable.

A **Reference** is called a **Pointer or a Memory address** in other languages. A reference is like your friend's address. The address is not your friend, but it's a way to reach your friend.

Primitive Data Types

In other languages, the format and size of primitive data types may depend on the platform on which the program is running.

In contrast, the java language specifies the size and format of its primitive data types hence, you don't have to worry about system dependencies.

Keywords	Description	Size
(Integers)		
`byte`	Byte-length integer	8-bit two's complement (1 byte)
`short`	Short integer	16-bit two's complement (2 byte)
`int`	Integer	32-bit two's complement (4 byte)
`long`	Long integer	64-bit two's complement (8 byte)
(Real numbers)		

float	Single-Precision floating point	32-bit (4 byte)
double	Double-precision floating point	64-bit (8 byte)
(Other types)		
char	A single character	16-bit Unicode character (2 byte)
boolean	A Boolean value	true or false

2.3.3 Identifiers

A name can be specify to memory location like variables, constants, array name, class name, object etc.,

Rules for Constructing Identifier Names

(i) The **Identifier** name must start with an alphabet, underscore and DollerSign and can contain only alphabets, number, underscore and DollerSign($).

(ii) No space between **Identifier** names.

(iii) Java Keywords cannot be used as **Identifier** names.

(iv) Java is a case-sensitivity, an **Identifier** name with upper case letters is different from the same name with lower case letters.

2.3.4 Variables

A value that may change during the execution of a program.

Java allocates memory to each variable of a program. Each variable that is used must be declared and initialize it, because java does not have garbage value for variable When we create an object of a class, java assigns default-values to the primitive data type class variables.

Note Java dose not have Garbage Values for variables

Default Values

Data Type	Default Values
byte	0
short	0
int	0
long	0l
float	0.0f
double	0.0
char	'\u000'(null)
boolean	false
String	null

In addition to the name and type that you explicitly given to a variable, a variable also has scope.

Scope of a Variable It is the life span of the variable it determines what sections of code can use the variable. The location of the variable declaration, that is where the declaration appears in relation to other code elements, determines its scope.

Local	Global
Variables that are declared in a method are called local variables. They are not accessible outside the method, which they are declared.	Variables that are declared outside method are called global variables. They are accessible to all methods.

2.3.5 Literals or Constants

A value that does change during the execution of a program

Example

Integer	15,-15,28
Real	15.0,25.89,0.028
Character	‘k’,’$’,’5’
String	"palar","128","#$%"
Boolean	true, false

Declaration of Variables and Constants

Syntax

```
datatype <identifierName>;
```

Example

```
float  radius; //(variable)

final float PI=3.14f; //(constant)
```

Note : Declaration of variables or constants anywhere in the program.

Naming Convention

Class Name	Each Word of a First character begins with an **Uppercase letter**.	**Eg:** Student, StudentDetails
Variable or **Method**	First word of character begins **with a lowercase letter** each word after the first word begins with an uppercase letter.	**Eg:** If names have more than one word, such as **isVisible(), parseInt()**
Constant	Define as fully uppercase	**Eg:** PI