✓ History of JAVA:

- Java was concived (imagine) by James Gosling, Patrick Naughton, Chris Warth, Ed frank, and Mike Sheridan at sun Microsystems (subsidiary of Oracle Corporation) in 1991.It took 18 month to develop the first working version.

- First it was called “Greentalk” by james gosling and the file extension was .gt

- After that it was called “OAK” and was developed as a part of green project.
- **Why oak?**: Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
- Oak was renamed as "Java" In 1995 because it was already a trademark by Oak Technologies
- **Why JAVA?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say. According to James Gosling, "Java was one of the top choices along with Silk". Since Java was so unique, most of the team members preferred Java than other names. (silk is a set of java classes that enable objected oriented simulation and animation)
- Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.(espresso is a concentrated form of coffee)
- Java is name, not an acronym.
- Java highlight in 1995 as Time magazine called Java one of the Ten Best Products of 1995.
- JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.
- Java latest version is:
- The similarities between java and c++ . It is tempting to think of java as simply the “internet version of c++”. however to do so would be a large mistake. Java has significant practical and philosophical differences.
- Java was influenced(effect) by c++ it is not enhanced version of c++. example: java is neither upwardly nor downwardly compatible with c++.
- Java enhances and refines the object oriented paradigm(model) used by c++.

✓ Application of java:

- Application is a program that runs on your computer, under the operating system of that computer.
- Java provides a rich and wide range of API that helps programmers to develop applications. Using Java, we can develop different applications for different purposes. We can use Java technology to develop the following applications:

(1) **Desktop GUI Applications**: User can also develop a GUI application using Java. Java provides AWT, JavaFX, and Swing for developing the GUI based desktop application. The tools contain the pre-assembled components like list, menu, button.

(2) Mobile App Development: The Java programming language can be considered as the official language for mobile application development. Most of the android applications build using Java. The most popular android app development IDE Android Studio also uses Java for developing android applications. So, if you are already familiar with Java, it will become much easier to develop android applications. The most popular android applications Spotify and Twitter are developed using Java.

(3) Web-based Applications: It is also used for developing the web-based application because it provides vast support for web development through Servlet, JSP, and Struts. It is the reason that Java is also known as a server-side programming language. Using these technologies, we can develop a variety of applications. The most popular frameworks Spring, Hibernate, Spring Boot, used for developing web-based applications. LinkedIn, AliExpress, web.archive.org, IRCTC, etc. are the popular websites that are written using Java programming language.

(4) Game Development: Java is widely used by game development companies because it has the support of the open-source most powerful 3D engine.

(5) Big Data Technology: As many programming languages are available for Big Data Technology but still Java is the first choice for the same. The tool Hadoop HDFS platform for processing and storing big data applications is written in Java. In big data, Java is widely used in ETL applications such as Apache Camel and Apache Kafka. It is used to extract and transform data, and load in big data environments**.**

(6) Distributed Applications: The JINI (Java Intelligent Networking Infrastructure) provides the infrastructure to register and find distributed services based on its specification. It implements a mechanism that is known as JavaSpaces. It supports the distribution, persistence, and migration of objects in a network.

(7) Cloud-Based Applications: A cloud application is the on-demand availability of IT resources via the internet. The cloud-based application provides the service at a low cost. Java provides the environment to develop cloud-based applications. We can use Java to develop SaaS (Software as a Service), LaaS (Logging as a Service), and PaaS (Platform as a Service). The cloud application widely used to share data between companies or to develop applications remotely.

(8) **IoT Application:** IoT is a technology that connects the devices in its network and communicates with them. IoT has found almost in all the small devices such as health gears, smartphones, wearables, smart lighting, TVs, etc. For developing the IoT application there is a lot of programming languages that can be used but Java offers an edge to developers that is unparalleled. IoT programmers gravitate towards Java because of its security, flexibility, and versatility.

→Compiler vs interpreter

Parameter	Compiler	Interpreter
Program scanning	Compilers scan the entire program in one go.	The program is interpreted/translated one line at a time.
Error detection	As and when scanning is performed, all the errors are shown in the end together, not line by line.	One line of code is scanned, and errors encountered are shown.
Object code	Compilers convert the source code to object code.	Interpreters do not convert the source code into object code.
Execution time	The execution time of compiler is less, hence it is preferred.	It is not preferred due to its slow speed. Usually, interpreter is slow, and hence takes more time to execute the object code.
Need of source code	Compiler doesn't require the source code for execution later.	It requires the source code for execution later.
Programming languages	Programming languages that use compilers include C, C++, C#, etc..	Programming languages that uses interpreter include Python, Ruby, Perl, MATLAB, etc.
Types of errors detected	Compiler can check syntactic and semantic errors in the program simultaneously.	Interpreter checks the syntactic errors only.
Size	Compiler are larger in size.	Interpreters are smaller in size.
Flexibility	Compilers are not flexible.	Interpreters are relatively flexible.
Efficiency	Compilers are more efficient.	Interpreters are less efficient.

→Java can be considered both a compiled and an interpreted language because its source code is first compiled into a binary byte-code. This byte-code runs on the Java Virtual Machine (JVM), which is usually a software-based interpreter.

✓ **Bytecode**:

- Byte Code consisting of binary, hexadecimal, macro instructions like (new, add, swap, etc.) and it is not directly understandable by the CPU. It is designed for efficient execution by software such as a virtual machine. intermediate-level
- Byte code is a non-runnable code generated after compilation of source code and it relies on an interpreter to get executed.
- Byte code is considered as the intermediate-level code.
- Byte code is a non-runnable code generated after compilation of source code and it relies on an interpreter to get executed.
- Byte code is executed by the virtual machine then the Central Processing Unit.
- It is platform-independent as it is dependent on the virtual machine and the system having a virtual machine can be executed irrespective of the platform.

✓ JVM (java virtual machine):

- Bytecode is a highly optimized set of instructions designed to be executed by the java run-time system, which is called the java virtual machine (JVM).
- Jvm is an interpreter for bytecode.
- Bytecode helps makes it much easier to run a program in a wide variety of environment. The reason is simple only the jvm needs to be implemented for each platform.
- Java program is interpreted also helps to make it secure because the execution of every java program is under the control of jvm, the jvm can contain the program and prevent it from generating side effects outside of the system.
- The use of bytecode enables the java runtime system to execute programs much faster than you might expect (Hope).
- Definition: JVM: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

about jvm:

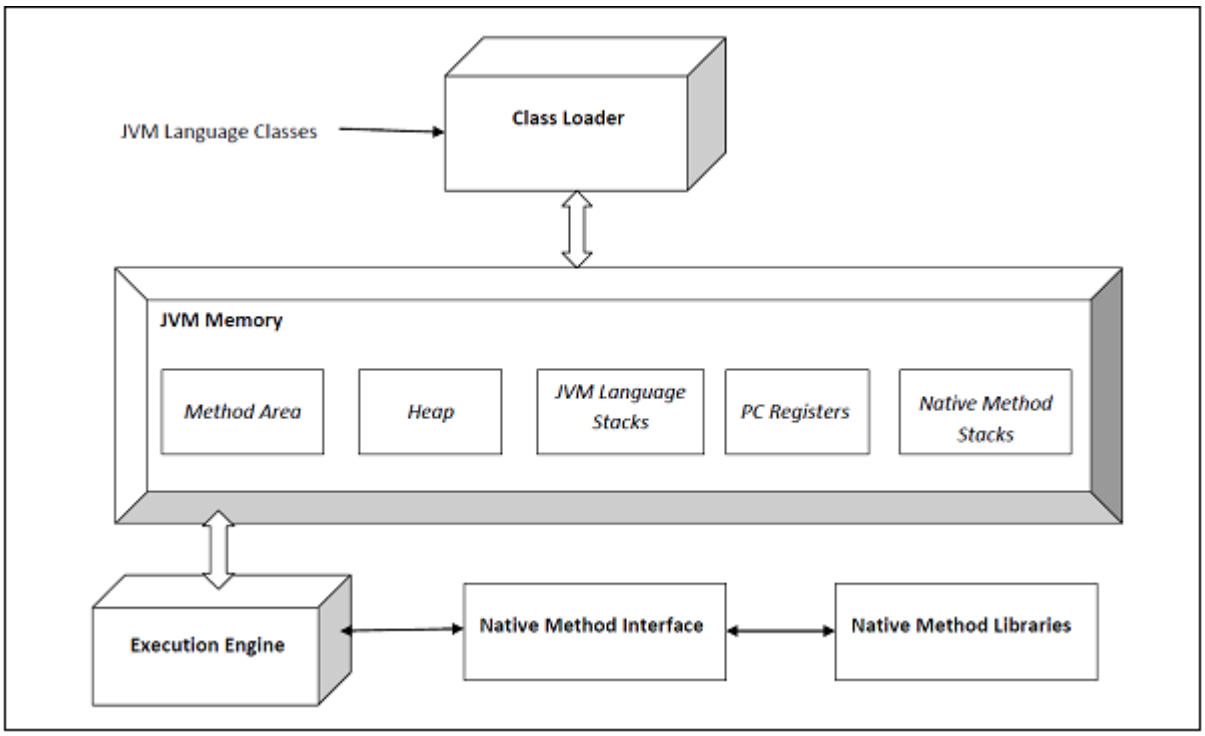
- **Its implementation has been provided by Oracle and other companies.**
- **Its implementation is known as JRE (Java Runtime Environment)**
- **you write java command on the command prompt to run the java class, an instance of JVM is created.**

✓ **JVM perform following operation:**

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

- ✓ **Jvm provide definition for the:**
  - Memory area
  - Class file format
  - Register set
  - Garbage-collected heap
  - Fatal error reporting etc.

- ✓ **JVM ARCHITECTURE:**



→main components of jvm architecture

(1) Class Loder:

- ✓ The JVM manages the process of loading, linking and initializing classes and interfaces in a dynamic manner. During the loading process, the JVM finds the binary representation of a class and creates it.
- ✓ During the linking process, the loaded classes are combined into the run-time state of the JVM so that they can be executed during the initialization phase. The JVM basically uses the symbol table stored in the run-time constant pool for the linking process. Initialization consists of actually executing the linked classes.

The following are the types of class loaders:

- **BootStrap class loader**: This class loader is on the top of the class loader hierarchy. It loads the standard JDK classes in the JRE's *lib* directory.
- **Extension class loader**: This class loader is in the middle of the class loader hierarchy and is the immediate child of the bootstrap class loader and loads the classes in the JRE's lib\ext directory.
- **Application class loader**: This class loader is at the bottom of the class loader hierarchy and is the immediate child of the application class loader. It loads the jars and classes specified by the **CLASSPATH ENV** variable.

2. Linking and Initialization

The linking process consists of the following three steps −

- ✓ Verification − This is done by the Bytecode verifier to ensure that the generated .class files (the Bytecode) are valid. If not, an error is thrown and the linking process comes to a halt.
- ✓ Preparation − Memory is allocated to all static variables of a class and they are initialized with the default values.
- ✓ Resolution − All symbolic memory references are replaced with the original references. To accomplish this, the symbol table in the run-time constant memory of the method area of the class is used.

Initialization is the final phase of the class-loading process. Static variables are assigned original values and static blocks are executed.\

3.Runtime Data Areas

The JVM spec defines certain run-time data areas that are needed during the execution of the program. Some of them are created while the JVM starts up. Others are local to threads and are created only when a thread is created (and destroyed when the thread is destroyed). These are listed below −

- ✓ **PC (Program Counter) Register**

It is local to each thread and contains the address of the JVM instruction that the thread is currently executing.

- ✓ **Stack**

It is local to each thread and stores parameters, local variables and return addresses during method calls. A StackOverflow error can occur if a thread demands more stack space than is permitted. If the stack is dynamically expandable, it can still throw OutOfMemoryError.

- ✓ **Heap**

It is shared among all the threads and contains objects, classes' metadata, arrays, etc., that are created during run-time. It is created when the JVM starts and is destroyed when the JVM shuts down. You can control the amount of heap your JVM demands from the OS using certain flags (more on this later). Care has to be taken not to demand too less or too much of the memory, as it has important performance implications. Further, the GC manages this space and continually removes dead objects to free up the space.

✓ **Method Area**

This run-time area is common to all threads and is created when the JVM starts up. It stores per-class structures such as the constant pool (more on this later), the code for constructors and methods, method data, etc. The JLS does not specify if this area needs to be garbage collected, and hence, implementations of the JVM may choose to ignore GC. Further, this may or may not expand as per the application's needs. The JLS does not mandate anything with regard to this.

✓ **Run-Time Constant Pool**

The JVM maintains a per-class/per-type data structure that acts as the symbol table (one of its many roles) while linking the loaded classes.

✓ **Native Method Stacks**

When a thread invokes a native method, it enters a new world in which the structures and security restrictions of the Java virtual machine no longer hamper its freedom. A native method can likely access the runtime data areas of the virtual machine (it depends upon the native method interface), but can also do anything else it wants.

4. Execution Engine

The execution engine is responsible for executing the bytecode, it has three different components:

Garbage Collection

The JVM manages the entire lifecycle of objects in Java. Once an object is created, the developer need not worry about it anymore. In case the object becomes dead (that is, there is no reference to it anymore), it is ejected from the heap by the GC using one of the many algorithms – serial GC, CMS, G1, etc.

During the GC process, objects are moved in memory. Hence, those objects are not usable while the process is going on. The entire application has to be stopped for the duration of the process. Such pauses are called 'stop-the-world' pauses and are a huge overhead. GC algorithms aim primarily to reduce this time.

Interpreter

The interpreter Interprets the bytecode. It interprets the code fast but it's slow in execution.

JIT Complier

The JIT stands for Just-In-Time. The JIT compiler is a main part of the Java runtime environment and it compiles bytecodes to machine code at runtime.

5. **Java Native Interface (JNI)**

Java Native Interface (JNI) interacts with the native method libraries which are essential for the execution.

6. Native Method Libraries

Native method libraries are the collection of C and C++ libraries (native libraries) which are essential for the execution.

System .out. println:

System: System is a certain entity (class) that provides the developer with the ability to communicate with his environment
Out: outis the name of a variable that stores a reference to an object (entity) of type PrintStream
Println: It can take numbers, strings, and other objects as arguments;

- ✓ **Buzzwords in java**: The features of Java are also known as Java buzzwords.

(1) **simple**: Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

→Java syntax is based on C++ (so easier for programmers to learn it after C++).
→Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
→There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

(2) **Object-oriented:** Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
Basic concepts of OOPs are:
→Object
→ Class
→Inheritance
→Polymorphism
→Abstraction
→Encapsulation

(3) **Platform Independent**: Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

- ✓ There are two types of platforms software-based and hardware-based. Java provides a software-based platform.
- ✓ The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1.Runtime Environment
2. API (Application Programming Interface)

- ✓ Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA)

(4) **Secured**: Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

→No explicit pointer
→Java Programs run inside a virtual machine sandbox

→ **Classloader**: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
→**Bytecode Verifier**: It checks the code fragments for illegal code that can violate access rights to objects.
→**Security Manager**: It determines what resources a class can access such as reading and writing to the local disk.

(5) **Robust(strong):** Java is robust because:

→It uses strong memory management.
→There is a lack of pointers that avoids security problems.
→Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
→There are exception handling and the type checking mechanism in Java. All these points make Java robust.

(6) **Architecture-neutral**: Java is architecture neutral because java virtual machine ability to "write once, run anywhere, anytime, forever"
(7) **Portable**: Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
(8) **High-performance**: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.
(9) **Distributed**: Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a File. The original version of java included features for intra- address-space messaging. This allowed objects on two different computers to execute procedures remotely. Java revived these interfaces in a package called Remote method Invocation (RMI). This feature Brings an unparalleled level of abstraction to client/server programming.
(9) **Multithreaded**: Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct

smoothly running interactive systems. Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem.
(10) **Dynamic**: Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

→c++ vs java

Sr.No	Comparision	C++	Java
1	**Platform-independent**	C++ is platform-dependent.	Java is platform-independent.
2	**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.
3	**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.
	Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer

			support in java.
	Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
	Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value and call by reference is done through passing an object in java.
	Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
	Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
	Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
	Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
	unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >>

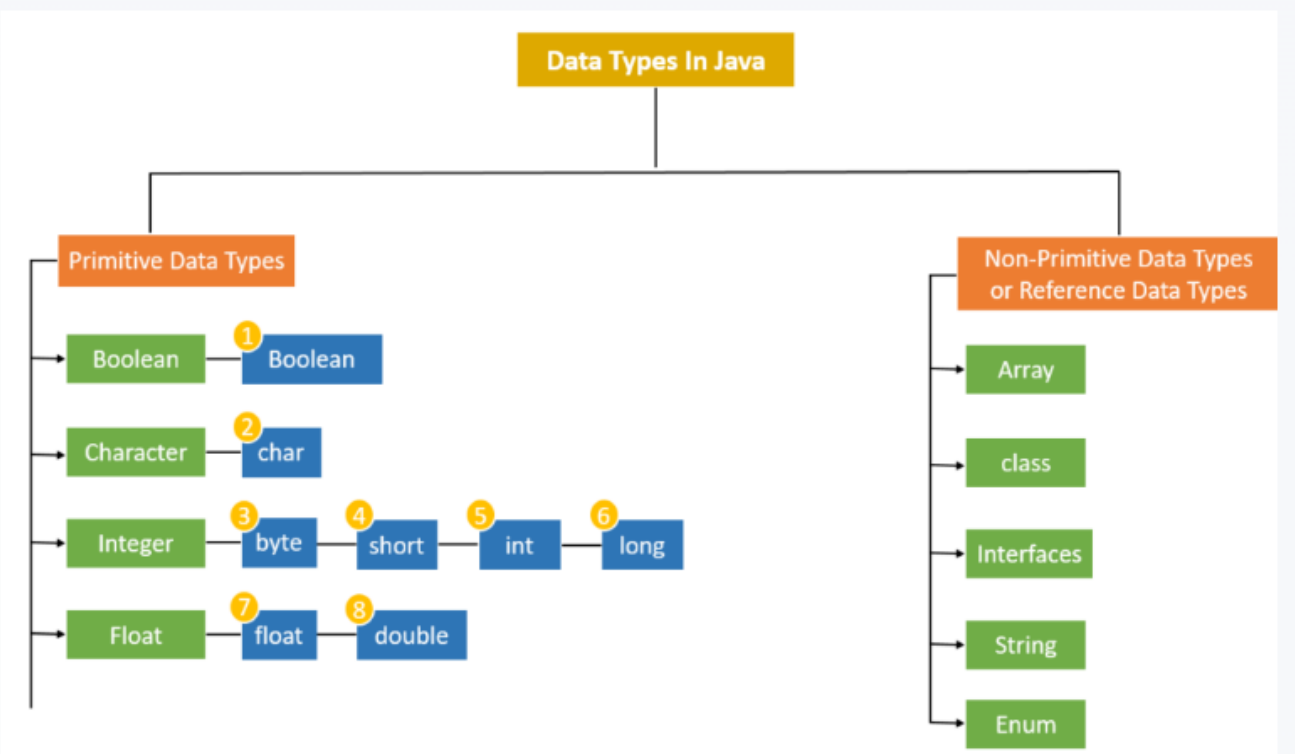
			operator.
	Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
	Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
	Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java. lang. Object.

Features	**C++**	**Java**
Abstraction	Yes	Yes
Encapsulation	Yes	Yes
Single Inheritance	Yes	Yes
Multiple Inheritance	Yes	No
Polymorphism	Yes	Yes
Static Binding	Yes	Yes
Dynamic Binding	Yes	Yes

Features	C++	Java
Operator Overloading	Yes	No
Header Files	Yes	No
Pointers	Yes	No
Global Variables	Yes	No
Template Class	Yes	No
Interference and Packages	No	Yes
API	No	Yes

→**Data Types:** Data types specify the different sizes and values that can be stored in the variable. There are main two types of data types in Java

1) **Primitive data types**: The primitive data types include Boolean, char, byte, short, int, long, float and double
2) **Non-primitive data types**: The non-primitive data types include Classes, Interfaces, and Arrays



Data Type	Description
byte	Stores whole numbers from -128 to 127
short	Stores whole numbers from -32,768 to 32,767
int	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	Stores fractional numbers. Sufficient for storing 15 to 16 decimal digits
boolean	Stores true or false values
char	Stores a single character/letter or ASCII values

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

JAVA: Java is a high level, robust, object-oriented and secure programming language.
→ Java was developed by Sun Microsystems (which is now the subsidiary (additive or supplement) of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

It is used for:

1. Mobile applications (especially Android apps)
2. Desktop applications
3. Web applications
4. Web servers and application servers
5. Games
6. Database connection

Use of java
→Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
→It is one of the most popular programming languages in the world
→It has a large demand in the current job market
→It is easy to learn and simple to use
→It is open-source and free
→It is secure, fast and powerful
→It has huge community support (tens of millions of developers)
→Java is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
→As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

Advantages of Java

1. Java is an Object-Oriented Programming Language
2. Simple Syntax for Easier Learning
3. Created the Standard for Enterprise Computing
4. Extensive Talent Pool
5. Java is a Secure Language
6. Java is a Distributed Language
7. Platform-Independent (Write Once Run Anywhere!)
8. Java Provides Automatic Memory Management
9. Java has Massive Community Support
10. Java supports Multithreading

Disadvantages of java

- High Level of Code Complexity
- Java programs take much longer time to run compared to C/C++.
- it consumes more memory.
- There is no support for low level programming in Java, like pointers are missing.
- There is no control over garbage collection in Java. That is programmer does not have any right to control the garbage collection. Java does not provide functions like delete(), free().

Operators:

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result

Java operators are symbols used to execute operations on variables and operate on the values of the operands

Operator Type	Category	Precedence	Associativity
Unary	postfix	a++, a--	Right to left
	prefix	++a, --a, +a, -a, ~, !	Right to left
Arithmetic	Multiplication	*, /, %	Left to Right
	Addition	+, -	Left to Right
Shift	Shift	<<, >>, >>>	Left to Right
Relational	Comparison	<, >, <=, >=, instanceOf	Left to Right
	equality	==, !=	Left to Right
Bitwise	Bitwise AND	&	Left to Right
	Bitwise exclusive OR	^	Left to Right
	Bitwise inclusive OR	|	Left to Right
Logical	Logical AND	&&	Left to Right
	Logical OR	||	Left to Right
Ternary	Ternary	? :	Right to Left
Assignment	assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Right to Left

→**Basic concept of oop:** Object-Oriented Programming is a paradigm(model) that provides many concepts, such as class and object, inheritance, data binding, polymorphism, etc.

Object → Class → Inheritance → Polymorphism → abstraction → Encapsulation

(1) Object: Any entity that has state and behavior is known as an object

(2) Class: A class is a user defined data type which hold data items and member functions OR Collection of objects is called class. It is a logical entity.

(3) Inheritance: one class can achieve all the properties and methods of another class is known as Inheritance

(4) Polymorphism: one task is performed in different ways, it is known as polymorphism. OR polymorphism is the ability of an entity to have more than one form

(5) Abstraction: Hiding internal implementation and showing functionality only to the user is Known as abstraction.
→ Java, we use abstract class and interface to achieve abstraction.

(6) Encapsulation: Binding (or wrapping) code and data together into a single unit are known As encapsulation
→ A Java class is the example of encapsulation because all the data members are private here

→**Structure of java program:**

Documentation Section
Package Statement
Import Statement
Interface Statement
Class Definition

Main Method Class
{
//Main method defintion
}

Documentation Section:

- It includes the comments that improve the readability of the program.
- It consists comments in Java that describe the purpose of the program, author name, date and time of program creation.
- This section is optional and comments may appear anywhere in the program.
  Java programming language supports three types of comments..
  - Single line Comment (or end-of line comment)
    - It starts with a double slash symbol (//) and terminates at the end of the current line.

    Ex: // sum of two numbers
  - Multi-line Comment

    /* a multi-line comment is declared like this

    and can have multiple lines as a comment */
  - Documentation Comment
    - /** a documentation comment starts with a delimiter and ends with */

Package Statement

- A package is a collection of classes, interfaces and sub-packages.
- A sub package contains collection of classes, interfaces and sub-sub packages etc.
- There is a provision in Java that allows you to declare your classes in a collection called package.
- There can be only one package statement in a Java program.
- It must appear as the first statement in the source code file before any class or interface declaration.
- This statement is optional,

Syntax: package package_name;

Example: package student;

- This statement declares that all the classes and interfaces defined in this source file are a part of the student package.

  java.lang.*;

  package is imported by default and this package is known as default package.

Import Statement

- Many predefined classes are stored in packages in Java, an import statement is used to refer to the classes stored in other packages.
- An import statement is always written after the package statement but it has to be before any class declaration.
- You can import a specific class or all the classes of the package.
- Many classes can be imported in a single program and hence multiple import statements can be written.

```
import java.util.Date; //imports the date class
import java.applet.*; //imports all the classes from the java applet package
```

Interface Section

- This section is used to specify an interface in Java.
- It is an optional section which is mainly used to implement multiple inheritance in Java
- An interface is similar to a class but contains only constants and method declarations.
- An Interfaces cannot be instantiated.
- They can only be implemented by classes or extended by other interfaces.

```
Interface shape{
void  draw(int length, int bredth);
void  show();
}
```

Class Definition

- Java program may contain multiple class definition.
- Classes are primary feature of Java program.
- The classes are used to map real world problems.
- classes defines the information about the user-defined classes in a program.
- A class is a collection of variables and methods that operate on the fields.
- Every program in Java will have at least one class with the main method.

```
class Addition
{
    void add(String args[])
    {
      int a=2, b=3, c;
       c=a+b;
       System.out.println(c);
    }
}
```

Main Method Class

- Execution of a Java application starts from the main method.
- In other words, its an **entry point for the class** or **program** that starts in **Java Run-time**.
- The main () method which is from where the execution of program actually starts and follow the statements in the order specified.
- The main method can create objects, evaluate expressions, and invoke other methods and much more.
- On reaching the end of main, the program terminates and control passes back to the operating system.
- The class section is mandatory.

```
// Program to display message on the screen
 class HelloJava {
public static void main(String args[])
 {
   System.out.println("Hello  Harsha");
 }}
```

→ java command line arguments:

- ✓ Java command line arguments is a way to pass inputs to the java program during application execution
- ✓ Command line arguments can be passed by multiple ways to Java application or program. Most commonly, command line arguments are passed from console where a java program is executed.
- ✓ command-line arguments, provided during program execution, are captured in the main() method as a string array.
- ✓ user can pass any number of command line arguments to java program.
- ✓ The Java Virtual Machine (JVM) encapsulates these inputs into the args [] array. We can check the number of arguments passed using args. length. In case no command line argument is present, then this array is empty.
- ✓ user can pass Strings, integers and any other primitive value using command line arguments. Each argument passed is available in the array in the order of entry, starting from args [0].
- ✓ Command line arguments facilites user input retrieval and manipulation in case of console based applications.

  - **Example:** single command line argument.
  - User are checking if exactly one argument is passed to represent name. In case, no arguments are passed or more than one argument is passed, we print the error message as invalid number of arguments passed. Otherwise, we're printing the name with a salutation**.**

```
public class excommand
{

  // args array represents the command line arguments passed
  public static void main(String[] args)
{
    // if only one argument is passed
    if(args.length == 1)
{
            String name = args[0];
      System.out.println("Welcome " + name + "!");
    }
else
{ // otherwise print an error message
      System.out.println("Invalid Command line argument(s) passed.");
 }
  }
}
```

Let us compile and run the above program without any command line argument, this will produce the following result −

```
D:\test>javac excommand.java

D:\test>java excommand
```

Let us compile and run the above program without any command line argument, this will produce the following result −

D:\test>javac excommand.java

D:\test>java excommand
Invalid Command line argument(s) passed.
Here, we've compiled the java code using javac command and then run using java command without any command line argument. Let's run the java command again with required parameter.

D:\test>java excommand java
Welcome java

- Example2

```
public class excommand2
{
          // args array represents the command line arguments passed
        public static void main(String[] args)
        {
      // if two arguments are passed
      if(args.length == 2) {
String name = args[0];
        // parse the age as int
        int age = Integer.parseInt(args[1]);
        System.out.println("Name: " + name + ", age: " + age);
      }else { // otherwise print an error message
        System.out.println("Invalid Command line argument(s) passed.");
      }
    }
}
```

D:\test>java excommand2 java 604

Name: java, age: 604

- Java command-line arguments are very useful to create parameterized java programs which can accept the parameters dynamically. Users have runtime control over the behavior of the program as arguments can be passed to the main() method. With command line arguments, we can manage program's output, set setup parameters, and specify input files at runtime without any compile time dependencies.

- ✓ **Array:** An array is a fixed-size sequential collection of elements of the same data type.
- ✓ Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** Arrays have a fixed size and do not grow dynamically at runtime.

Array Type

(1) Single - dimensional (one dimensional)
(2) Multi - Dimensional

Declaration of array:

- ✓ One-dimensional:
  - Int a1[] = new int[3];
  - Int[] a2 = new int[3];
  - int a[]={1,2,3,4};
- ✓ Multi-dimensional:
  - Char twod1[][]=new char[3][4];
  - Char[][] two2=new char[3][4];

➢ **Example of single-dimensional array:** java program input 10 numbers in an array and display only the even numbers if present in the array.

```
import java.util.Scanner;

public class Example
{
   public static void main(String args[])
   {
      int a[]=new int[10], i;
      Scanner sc=new Scanner(System.in);

      System.out.println("Enter 10 numbers");
      for(i=0; i<10; i++)
      {
         a[i]=sc.nextInt();
      }

      System.out.println("List of even numbers");
      for(i=0; i<10; i++)
      {
         if(a[i]%2==0)
         {
            System.out.print(a[i]+" ");
         }
      }
   }
}
```

- Example of multi-dimensional array: Program to input numbers in a 3x3 Matrix and display it.

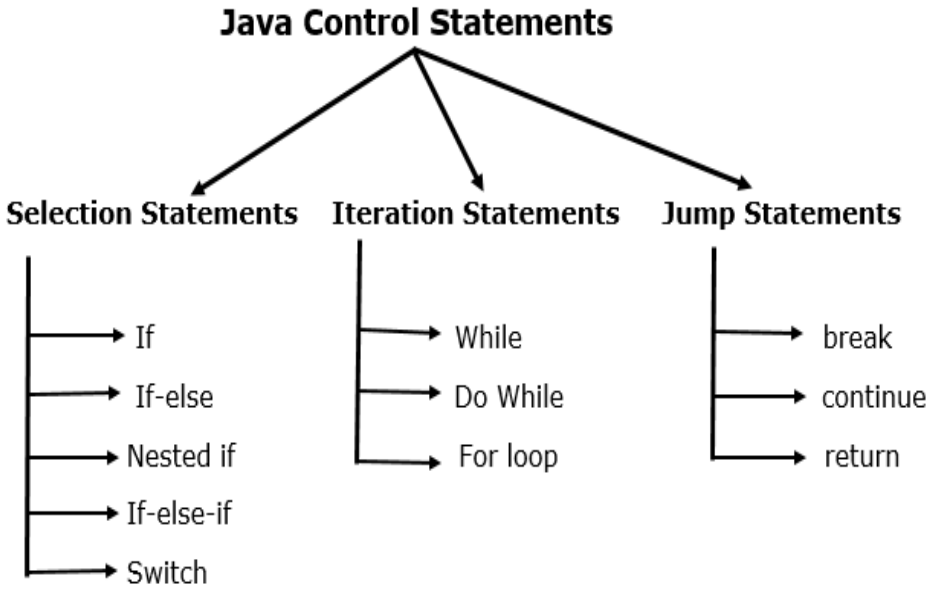
```
import java.util.Scanner;

public class Example
{
   public static void main(String args[])
   {
      int a[][]=new int[3][3];
      Scanner sc=new Scanner(System.in);
      int r,c;

      System.out.println("Enter 9 numbers");
      for(r=0; r<3; r++)
      {
         for(c=0; c<3; c++)
         {
            a[r][c]=sc.nextInt();
         }
      }

      System.out.println("\nOutput");
      for(r=0; r<3; r++)                    // this loop is for row
      {
         for(c=0; c<3; c++)              // this loop will print 3 numbers in each row
         {
            System.out.print(a[r][c]+" ");
         }
         System.out.println();   // break the line after printing the numbers in a row
      }
   }
}
```

- ✓ **Control Statement in java:** Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements**.**
- ✓ Java provides main three types of control flow statements.



Control Flow Statements Type	Control Flow Statement	Description
Conditional Statements	**if-else**	Executes a block of code if a specified condition is true, and Executes another block if the condition is false.
	switch-case	Evaluates a variable or expression and executes code based on matching cases.
Looping Statements	**for**	Executes a block of code a specified number of times, typically iterating over a range of values.
	while	Executes a block of code as long as a specified condition is true.
	do-while	Executes a block of code once and then repeats the execution as long as a specified condition is true.
Jump Statements	**break**	Terminates the loop or switch statement and transfers control to the

Control Flow Statements Type	Control Flow Statement	Description
		statement immediately following the loop or switch.
	continue	Skips the current iteration of a loop and continues with the next iteration.
	return	Exits a function and returns a value to the caller.

- → **Scanner class:** Java Scanner class is part of the java. util package. It was introduced in Java 1.5 release. The Scanner is mostly used to receive user input and parse them into primitive data types such as int, double or default String
- → The first statement import java.util.Scanner; is mandatory when using Scanner class. Alternatively, You can also import Scanner like this: java.util.*, this will import all the classes present in the java.util package
- → While creating an object of Scanner class, we passed the System.in in the Scanner class constructor. This is to read the data from standard input.
- → methods of Scanner class.

Method	Description
nextInt()	**It reads an int value entered by the user**
nextFloat()	**It reads a float value entered by the user**
nextBoolean()	**It reads a boolean value entered by the user**
nextLine()	**It reads a line of text entered by the user**
next()	**This method reads a word entered by the user**
hasNextLine()	**Checks if there is another line of text entered by the user**
hasNextInt()	**Checks if there is another int value input by the user**
reset()	**It resets the scanner**

toString()	**It is used to get string representation of user input**
nextByte()	**This method reads a byte value entered by the user**
nextDouble()	**This method reads a double value entered by the user**
nextShort()	**It reads a short value entered by the user**
nextLong()	**It reads a long value entered by the user**