Q1. History of Java:

- Java was created in 1991 by James Gosling and his team at Sun Microsystems, evolving from a language for consumer electronics into a globally popular, platform-independent programming language.

- Java began in 1991 as part of the Green Project at Sun Microsystems, led by James Gosling, along with Mike Sheridan and Patrick Naughton.

- The goal was to develop a language that could run on a variety of digital devices, including TVs, set-top boxes, and remote controllers, emphasizing portability, security, and reliability.

- Initially, the language was called **GreenTalk**, then renamed **Oak** after an oak tree outside Gosling's office. Java was called Oak as it is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.

- Due to trademark issues, the team need to change the name and they wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

- In 1995, Oak was renamed as Java

    - Java is an island of Indonesia where first coffee was produced (called java coffee). In 1995, Time magazine called Java one of the Ten Best Products of 1995.

- The team eventually chose the name **Java**, inspired by Java coffee from Indonesia, reflecting the language's dynamic and energetic nature.

- Java is available as jdk and it is an open source s/w.

Java Platforms:

There are three main platforms for Java:

- Java SE (Java Platform, Standard Edition) – runs on desktops and laptops.
- Java ME (Java Platform, Micro Edition) – runs on mobile devices such as cell phones.
- Java EE (Java Platform, Enterprise Edition) – runs on servers.

Java Introduction:

- **Java** is a **high-level, class-based, object-oriented programming language** designed to be simple, secure, robust, portable, and platform-independent. It was developed by **James Gosling** and his team at **Sun Microsystems** and officially released in **1995**. Since **2010**, Java has been owned and maintained by **Oracle Corporation** following Oracle's acquisition of Sun Microsystems. Java follows the philosophy of "**Write Once, Run Anywhere**" **(WORA)**, which means that Java programs compiled into **bytecode** can run on any platform that has a compatible **Java Virtual Machine (JVM)** without requiring recompilation.

Java Versions:

Version	Codename	Release Date	Major Features Added
JDK 1.0	Oak	Jan 23, 1996	First official Java release, Applets, AWT, JVM, Security model.
JDK 1.1	Rebirth of Java	Feb 19, 1997	Inner Classes, JavaBeans, JDBC, RMI, Reflection, Event Delegation Model, AWT enhancements.
J2SE 1.2	Play ground	Dec 8, 1998	Swing GUI toolkit, Collections Framework, JIT Compiler, Java Plug-in, CORBA, strictfp keyword.
J2SE 1.3	Kestrel	May 8, 2000	HotSpot JVM, JavaSound API, JPDA (Java Platform Debugger Architecture), JNDI improvements.
J2SE 1.4	Merlin	Feb 6, 2002	Assertions (assert), Logging API, Regular Expressions, NIO, XML Processing, Image I/O, Preferences API, Cryptography enhancements.
J2SE 5.0	Tiger	Sep 30, 2004	Generics, Annotations, Autoboxing/Unboxing, Enhanced for-each loop, Enums, Varargs, Static Import, Concurrency Utilities.
Java SE 6	Mustang	Dec 11, 2006	JDBC 4.0, Compiler API, Scripting API, JAXB 2.0, Web Services, Performance improvements, JVM enhancements.
Java SE 7	Dolphin	Jul 28, 2011	Try-with-resources, Multi-catch, Diamond Operator (<>), Strings in switch, NIO.2, Fork/Join Framework, Binary Literals, Underscores in numeric literals.
Java SE 8	—	Mar 18, 2014	Lambda Expressions, Streams API, Functional Interfaces, Default & Static Interface Methods, Optional, Date & Time API (java.time), Nashorn JavaScript Engine.

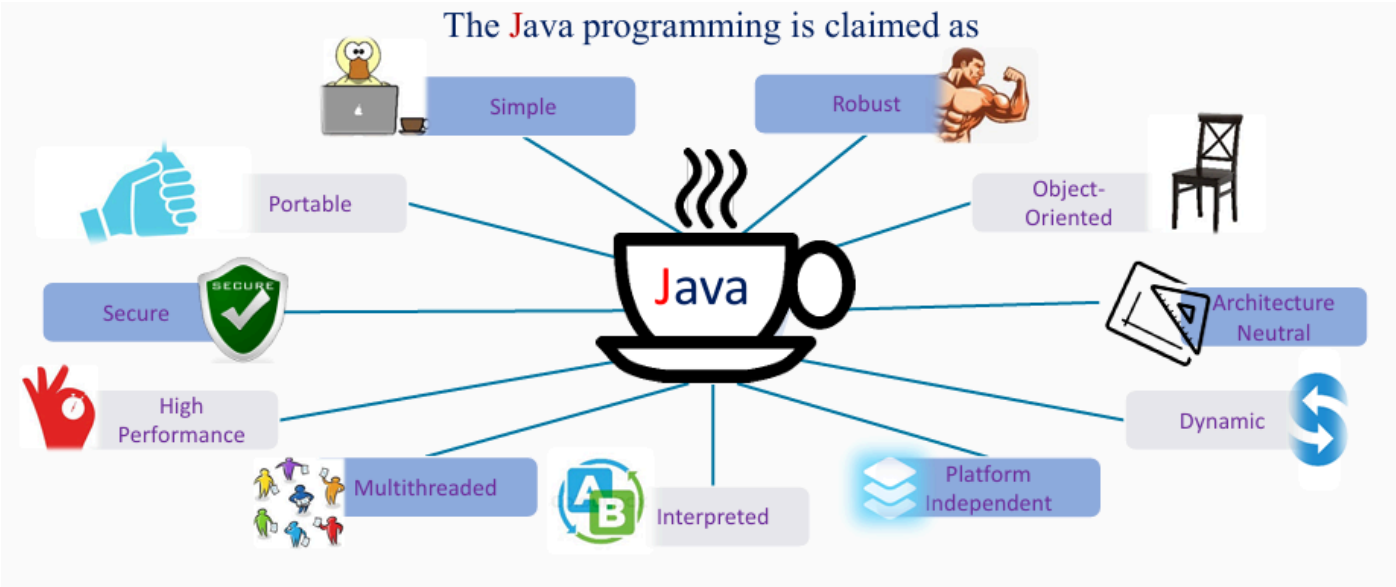
Version	Codename	Release Date	Major Features Added
Java SE 9	—	Sep 21, 2017	Module System (Project Jigsaw), JShell, Collection Factory Methods, Multi-release JARs, Process API improvements, Private Interface Methods.
Java SE 10	—	Mar 20, 2018	Local Variable Type Inference (var), Application Class-Data Sharing, Garbage Collector improvements.
Java SE 11 (LTS)	—	Sep 25, 2018	Standard HTTP Client, Single-file Source Code Execution, TLS 1.3, Flight Recorder, Removal of Java EE & CORBA modules.
Java SE 12	—	Mar 19, 2019	Switch Expressions (Preview), Shenandoah GC (Experimental), JVM improvements.
Java SE 13	—	Sep 17, 2019	Text Blocks (Preview), Dynamic CDS Archives, Socket API improvements.
Java SE 14	—	Mar 17, 2020	Switch Expressions (Standard), Records (Preview), Pattern Matching for instanceof (Preview), Helpful NullPointerExceptions.
Java SE 15	—	Sep 15, 2020	Text Blocks (Standard), Sealed Classes (Preview), Hidden Classes, Records (2nd Preview).
Java SE 16	—	Mar 16, 2021	Records (Standard), Pattern Matching for instanceof, Foreign Linker API (Incubator), Unix-domain Socket Channels.
Java SE 17 (LTS)	—	Sep 14, 2021	Sealed Classes, Pattern Matching enhancements, Strong Encapsulation, Foreign Function API (Incubator), New macOS rendering pipeline.
Java SE 18	—	Mar 22, 2022	UTF-8 by Default, Simple Web Server, Code Snippets in Javadoc, Vector API (Incubator).
Java SE 19	—	Sep 20, 2022	Virtual Threads (Preview), Record Patterns (Preview), Structured Concurrency (Incubator), Foreign Function & Memory API (Preview).
Java SE 20	—	Mar 21, 2023	Scoped Values (Incubator), Record Patterns (2nd Preview), Virtual Threads (2nd Preview), Structured Concurrency (2nd Incubator).
Java SE 21 (LTS)	—	Sep 19, 2023	Virtual Threads, Sequenced Collections, String Templates (Preview), Record Patterns, Pattern Matching for switch, Scoped Values (Preview), Generational ZGC.
Java SE 22	—	Mar 19, 2024	Foreign Function & Memory API, Statements before super(), String Templates (2nd Preview), Unnamed Variables & Patterns, Stream Gatherers (Preview), Class-file API (Preview).
Java SE 23	—	Sep 17, 2024	Module Import Declarations (Preview), Markdown Documentation Comments, Primitive Types in Patterns

Version	Codename	Release Date	Major Features Added
			(Preview), Class-file API improvements, Performance & JVM enhancements.
Java SE 24	—	Mar 18, 2025	Ahead-of-Time Class Loading improvements, Compact Object Headers (Experimental), Enhanced Pattern Matching, JVM and GC performance improvements.
Java SE 25 (LTS)	—	Sep 2025	Long-Term Support release, Stable enhancements to Project Loom, Foreign Function & Memory API refinements, Class-file API updates, Performance, Security and Garbage Collection improvements.
Java SE 26	—	Mar 2026	Latest feature release with continued improvements to Project Loom, Project Panama, JVM performance, garbage collectors, language previews, and developer productivity features.

Q2. Features or principles of Java:

Java was designed with several core principles that make it one of the most popular and reliable programming languages. The following are the core features of Java programming language

1. Simple to learn
2. Object Oriented
3. Platform independent
4. Architecture Neutral
5. Portable
6. Secured
7. Robust - (Strong)
8. Multithreaded
9. Distributed
10. High Performance
11. Dynamic
12. Interpreted and compiled



1. Simple to learn:

- Java is easy to learn and use.
- It has a clean syntax similar to C/C++ but removes complex features like pointers, operator overloading, and multiple inheritance through classes.
- Automatic memory management is provided through **Garbage Collection**.

2. Object-Oriented

- Java follows the **Object-Oriented Programming (OOP)** paradigm.
- Everything is organized into classes and objects.
- Supports the four main OOP principles:
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

3. Platform Independent

- Java programs are compiled into **bytecode** instead of machine code.
- Bytecode can run on any operating system that has a **Java Virtual Machine (JVM)**.
- This supports the **Write Once, Run Anywhere (WORA)** philosophy.

4. Architecture Neutral

- Java bytecode is independent of any specific processor architecture.

- The same compiled program can execute on different hardware platforms without modification.

5. Portable

- Java ensures consistent behavior across different platforms.
- Primitive data types have fixed sizes, and standard libraries behave uniformly.

6. Secure

- Java provides built-in security features such as:
  - No explicit pointers
  - Bytecode verification
  - Class Loader
  - Security Manager (legacy)
  - Exception handling
- These features help protect systems from malicious code.

7. Robust

- Java is designed to minimize runtime errors.
- Features contributing to robustness include:
  - Strong type checking
  - Automatic garbage collection
  - Exception handling
  - Memory management

8. Multithreaded

- Java supports **multithreading**, allowing multiple tasks to execute concurrently.
- This improves application performance and responsiveness.

9. Distributed

- Java provides libraries for developing distributed applications.
- APIs such as **RMI (Remote Method Invocation)**, sockets, and web services facilitate communication over networks.

10. High Performance

- Although Java is interpreted through the JVM, modern JVMs use the **Just-In-Time (JIT) Compiler** to convert frequently executed bytecode into native machine code, significantly improving performance.

11. Dynamic

- Java can load classes dynamically at runtime.
- It supports dynamic linking, reflection, and runtime object loading, making applications flexible and extensible.

12. Interpreted and Compiled

- Java source code is first **compiled** into bytecode by the Java compiler (javac).
- The JVM then **interprets** or **JIT-compiles** the bytecode into machine code for execution.

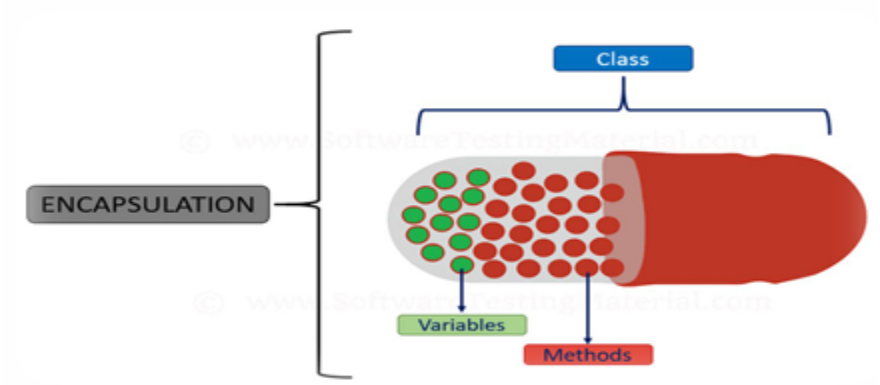
Q3. Java Programming paradigms:

Java is based on the concept of object-oriented programming. As the name suggests, at the center of it all is an object. Objects contain both data and the functionality that operates on that data. This is controlled by the following four paradigms

1. Encapsulation
2. Information hiding
3. Inheritance
4. Polymorphism

1. Encapsulation

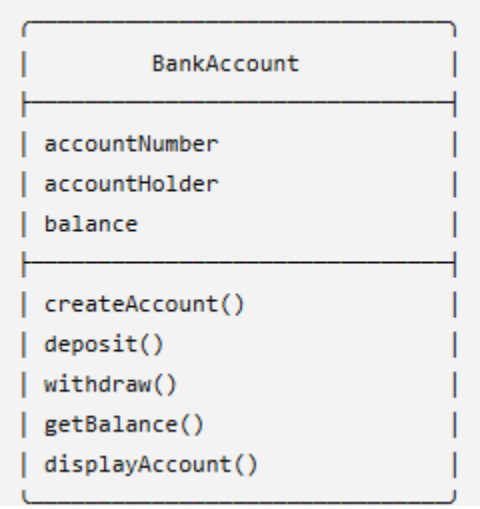
Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.



Example:

- The **data** (accountNumber, accountHolder, and balance) are encapsulated within the **BankAccount** class.
- Users cannot directly modify the data.
- They can access or update the data only through the **public methods** such as deposit (), withdraw (), and getBalance ().
- This combination of **data** and **operations** into a single unit is called **Encapsulation**.

- accountNumber and balance are private.
- Users cannot directly modify the balance.
- The balance can only be changed using methods like deposit().



Advantages of Encapsulation:

- **Data Security:** Prevents unauthorized access to data.
- **Data Hiding:** Internal implementation is hidden from the outside world.
- **Maintainability:** Changes to internal implementation do not affect external code.
- **Better Modularity:** Each class is self-contained and easier to understand.
- **Code Reusability:** Encapsulated classes can be reused in different applications.
- **Controlled Access:** Data can be validated before modification using methods such as getters and setters.

2. Information Hiding (Abstraction)

Information Hiding is the principle of **restricting direct access to the internal data and implementation details of a class**, exposing only the necessary functionality through public methods. It protects the object's data from unauthorized access and modification.

- Information hiding is achieved using access modifiers such as private, protected, and public.
- Data members are usually declared as private.
- Access to private data is provided through public methods (getters and setters).
- It prevents accidental or unauthorized modification of an object's data.
- It hides the internal implementation from the outside world.
- Users interact only with the interface (methods), not with the internal data.
- It improves security and data integrity.
- It makes the program easier to maintain because internal implementation can change without affecting other classes.
- It reduces the complexity of the system by exposing only essential information.
- Information hiding is one of the key benefits of Encapsulation.

Example:

In the above bank example

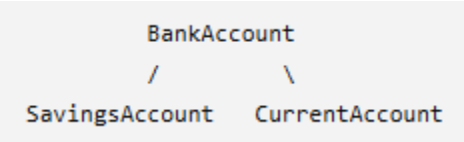
- accountNumber and balance are private.
- Users cannot directly modify the balance.
- The balance can only be changed using public methods like deposit ().

3. Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object-Oriented Programming system).

Example:

Inheritance allows one class to inherit the properties and methods of another class.

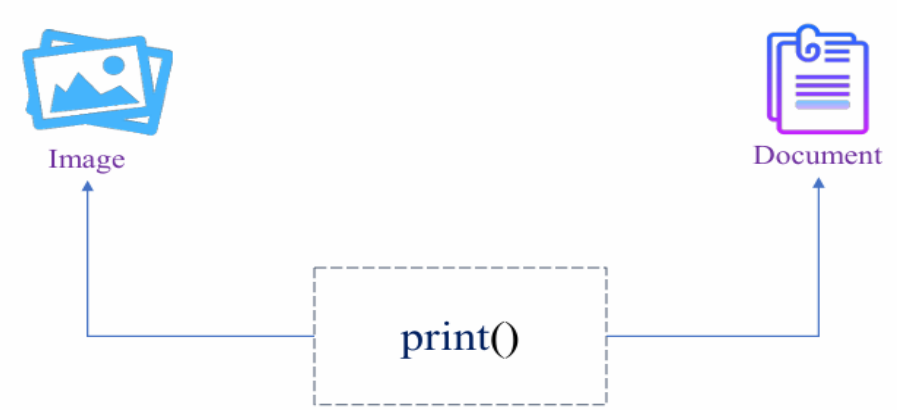


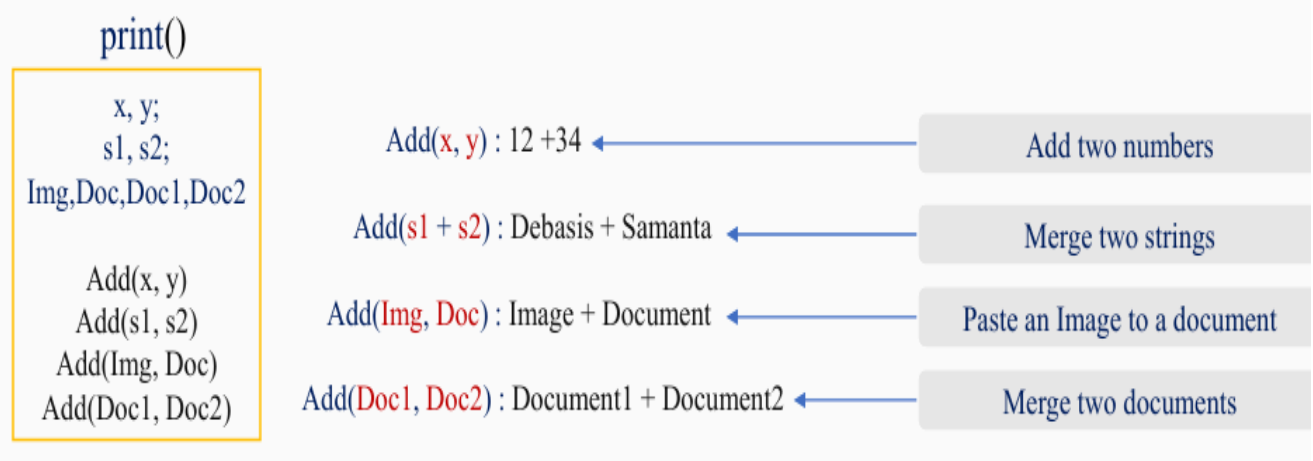
Both child classes SavingsAccoount and CurrentAccount inherit the public data & methods from BankAccount.

4. Polymorphism

In object-oriented programming, polymorphism refers to a programming language's ability to process objects depending on their class. In case of Bank example, the same method **calculateInterest ()** behaves differently depending on the account type. i.e., interest calculation method is different for SavingsAccoount and CurrentAccount.

If same task is performed in different ways based on which class object it is,

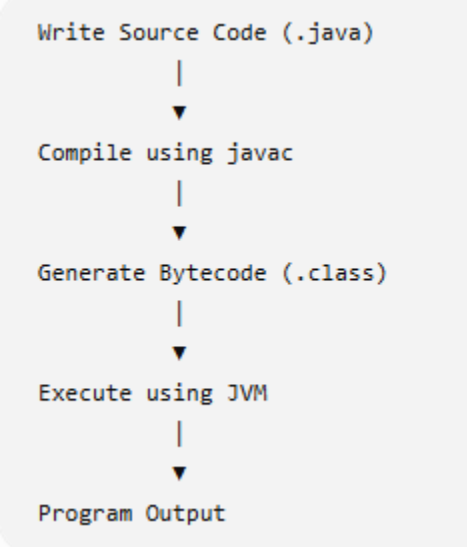




Q4. Differences between Procedure Oriented Programming and Object Oriented Programming.

Feature	Procedure-Oriented Programming (POP)	Object-Oriented Programming (OOP)
Basic Concept	Program is organized around functions or procedures.	Program is organized around objects and classes.
Approach	Top-down approach.	Bottom-up approach.
Focus	Focuses on functions and procedures.	Focuses on objects that contain data and methods.
Program Structure	Divided into functions.	Divided into classes and objects.
Data Handling	Data is shared globally among functions.	Data is encapsulated within objects.
Data Security	Less secure because data can be accessed directly.	More secure due to data hiding and encapsulation.
Encapsulation	Not supported.	Fully supported.
Abstraction	Not supported.	Supported using abstract classes and interfaces.
Inheritance	Not supported.	Supported for code reusability.
Polymorphism	Not supported.	Supported through method overloading and overriding.
Data Hiding	Not possible.	Possible using access modifiers (`private`, `protected`, `public`).
Reusability	Limited code reusability.	High code reusability through inheritance and composition.
Maintenance	Difficult to maintain for large applications.	Easier to maintain due to modular design.
Modularity	Less modular.	Highly modular.
Real-world Modeling	Does not model real-world entities effectively.	Models real-world entities using objects.
Communication	Functions communicate by passing data.	Objects communicate through method calls (messages).
Suitable For	Small and medium-sized applications.	Large, complex, and enterprise-level applications.
Examples of Languages	C, Pascal, FORTRAN	Java, C++, C#, Python, Ruby

Q5. How to edit compile and execute Java Program?



Step 1: Edit the Java Program

- Open any text editor or Integrated Development Environment (IDE) such as:
  - Notepad
  - Notepad++
  - Visual Studio Code
  - Eclipse
  - IntelliJ IDEA
  - NetBeans
- Write the Java source code.
- Save the file with the **.java** extension.
- The filename must be the same as the public class name.

Example:

```
public class HelloWorld {
   public static void main (String [] args) {
      System.out.println("Welcome to Java Programming!");
   }
}
```

Save the file as: HelloWorld.java

class: class keyword is used to declare classes in Java
public: It is an access specifier. public means this function is visible to all.

static: static is again a keyword used to make a function static. To execute a static function, you do not have to create an Object of the class. The main () method here is called by JVM, without creating any object for class.
void: It is the return type, i.e., this function will not return anything.
main: main () method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main () method. If a java class is not having a main () method, it causes compilation error.
System.out.println: This is used to print anything on the console like printf in C language.

Step 2: Compile the Java Program

The Java compiler (javac) translates the source code into **bytecode**.

```
javac HelloWorld.java
```

If there are **no syntax errors**, the compiler creates a bytecode file:

```
HelloWorld.class
```

If there are errors, the compiler displays error messages that must be corrected before recompiling.

Step 3: Execute (Run) the Java Program

The Java Virtual Machine (JVM) executes the compiled bytecode.

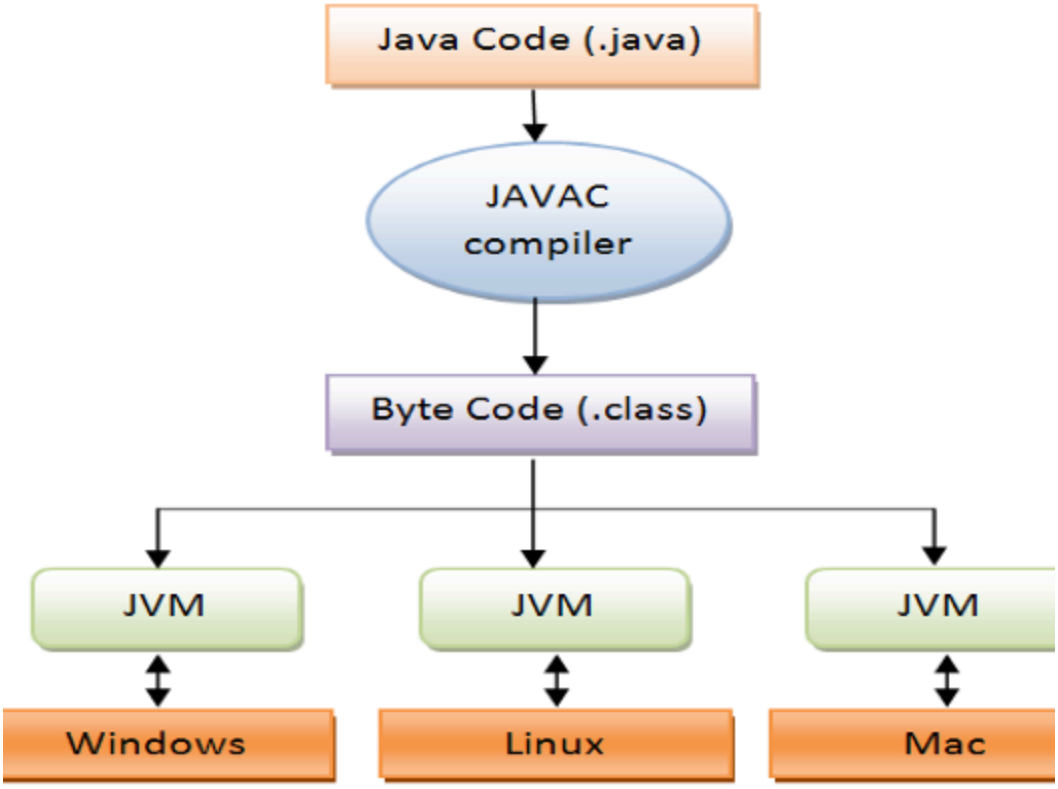
```
java HelloWorld
```

Note: Do **not** include the .class extension.

It produces the output

```
Welcome to Java Programming!
```

JVM:



Java complier translates the java source code into byte code or intermediate code, not the executable file. JVM take the byte code and convert into executable code corresponding to Operating system.

The output of a Java compiler is a bytecode. 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).

The difference between .exe file and .Class file is

- .exe file contains machine code understandable to the process. .exe file is System dependent, platform dependent.
- .class file contains byte code to understandable to the JVM, .class file is system independent, platform independent.

Q6. Java API:

- API (Application Programming Interface) The API enables Java programmers to develop varieties of applets and applications.
- It contains packages like

- **java. applet**– for applet programming
- **java.awt**– the Abstract Windowing Toolkit for designing GUI like Button, Checkbox, Choice, Menu, Pannel, etc.
- **java.io**– file input/output handling
- **java. lang**– provides useful classes like to handle Object, Thread, Exception, String, System, Math, Float, Integer, etc
- **java.net**– classes for network programming; supports TCP/IP networking protocols
- **java. util**– it contains miscellaneous classes like Vector, Stack, List, Date, Dictionary, Hash etc
- **javax. swing**– for designing graphical user interface (GUI)
- **java.sql**– for database connectivity (JDBC)

Q7. Java Coding conventions (or) Java Programming Style:

Java Programming style refers to a set of coding conventions and best practices followed while writing Java programs. These guidelines improve the **readability, maintainability, consistency, and quality** of the code.

Following a good programming style makes the code easier to understand, debug, and maintain by other programmers.

1. Use Meaningful Identifiers

Choose descriptive names for variables, methods, classes, and objects.

Good

```
int studentAge;
double accountBalance;
```

Bad

```
int a;
double x;
```

2. Follow Java Naming Conventions

Java is a case sensitive language. The rules to be followed by the programmers by writing class name, method name, and variable declaration etc. in the java programs.

There are six rules in the java naming conventions…

1. Package Names in java are written in all small letters.

   Ex: java.awt;
   java.io;
   javax. swing;

2. Class Names & Interface Names are start with capital letters.

   Ex: String

DataInputStream
ActionListene

3. Method Names start with a small letter, then each word start with capital letters.

   Ex: println ()
   readLine ()
   getNumberInstance ()

4. Variable also fallows the method Naming convention rule (i.e. Method name rules)
   Ex: empName
   Emp_Net_Sal
   cityName
5. Constant variable name should be written using all capital letters
   Ex: PI
   MAX_VALUE
   Font.BOLD
6. All Key words should be written in all small latters
   Ex: public
   void
   import

3. Proper Indentation

Indent the code consistently (usually **4 spaces**).

Good

```
if (marks >= 50) {
    System.out.println("Pass");
}
```

Bad

```
if(marks>=50){
System.out.println("Pass");
}
```

4. Use Proper Comments

Comments improve code readability.

Single-line Comment

```
// Calculate total marks
```

Multi-line Comment

```
/*
  This program calculates
  the average marks.
```

```
*/
```

Documentation Comment

```
/**
 * Displays student details.
 */
```

5. Keep Methods Small

Each method should perform **one specific task**.

Good

```
calculateTotal();
displayResult();
```

Instead of writing everything inside the main() method.

6. Declare Variables Close to Their Usage

Declare variables where they are needed.

Good

```
int total = marks1 + marks2;
```

Avoid unnecessary global variables.

7. Use Appropriate Access Modifiers

Protect data using access modifiers.

```
private int age;
public void display() {}
```

8. Avoid Magic Numbers

Instead of writing fixed values directly, use **constants**.

Bad

```
salary = salary + 5000;
```

Good

```
final int BONUS = 5000;
salary = salary + BONUS;
```

9. Write One Statement Per Line

Good

```
int a = 10;
int b = 20;
```

Bad

```
int a = 10; int b = 20;
```

10. Close Resources

Always close resources like Scanner, files, and database connections.

```
Scanner sc = new Scanner(System.in);
```

```
// Code

sc.close();
```

11. Avoid Duplicate Code

Reuse methods whenever possible.

```
calculateAverage ();
```

instead of writing the same code multiple times.

12. Follow Proper Package Structure

Organize classes into packages.

com. college. student
com. college. library

Q8. Elements or Tokens in JAVA:

Tokens are the **smallest individual units (building blocks)** of a Java program that have a meaningful interpretation to the compiler. it is used by the Java compiler for constructing expressions and statements. Every Java program is composed of a sequence of tokens.

Example

```
int age = 20;
```

The above statement contains the following tokens:

Token	Type
int	Keyword
age	Identifier
=	Operator
20	Literal
;	Separator

Types of Tokens in Java

Java tokens are classified into the following categories:

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Separators (Special Symbols)

1. Keywords

Keywords are **reserved words** that have predefined meanings in Java. They cannot be used as identifiers.

Java Keywords can not be used as a variable name.

Example:

Abstract	Assert	boolean	break	byte	case
catch	char	class	const	continue	Default
Do	double	else	enum	extends	final
finally	float	for	goto	if	implements
import	instanceOf	int	interface	long	natve
new	Package	private	Protected	public	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	void
volatile	while	true	false	null	

2. Identifiers

Identifiers are the names given to:

- Classes
- Objects
- Variables
- Methods
- Packages
- Interfaces

Rules for Naming Identifiers

1. Must begin with a letter, _ or $
2. Cannot begin with a digit
3. Cannot be a keyword
4. Case-sensitive
5. Can contain letters, digits, _, and $

Valid Identifiers

student
student1
_marks
salary

Invalid Identifiers

1student
class
total marks

3. Literals

Literals are constant values assigned to variables.

Types of Literals

Type	Example
Integer	100, 25
Floating-point	3.14, 2.5f
Character	'A', '9'
String	"Java"
Boolean	true, false
Null	null

Example:
```
int age = 20;
char grade = 'A';
String name = "John";
boolean result = true;
```

4. Operators

Operators perform operations on operands.

Operator Type	Examples
Arithmetic	+ - * / %
Relational	== != > < >= <=
Logical	&& ||
Assignment	= += -= *= /=
Increment/Decrement	++ --
Bitwise	& | ^ ~
Shift	<< >> >>>
Conditional (Ternary)	?:
Instance	instanceof

5. Separators (Special Symbols)

Separators separate different parts of a Java program.

Separator	Purpose
( )	Method calls, expressions
{ }	Block of code
[ ]	Arrays
;	End of statement
,	Separate variables/parameters
.	Access members of an object

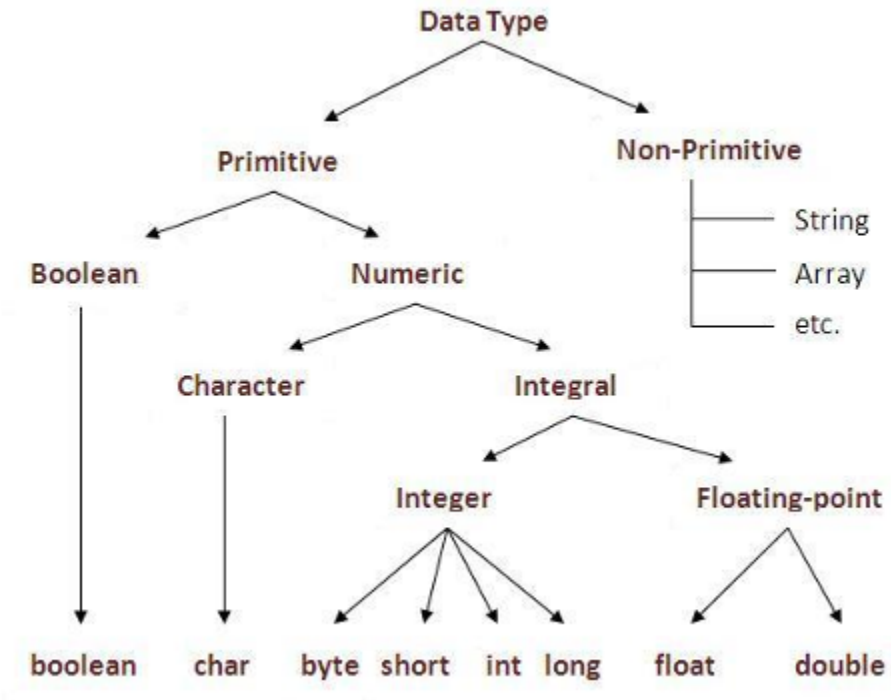
Summary Table of Tokens in Java:

Token	Description	Example
Keywords	Reserved words with predefined meaning	class, int, if, return
Identifiers	Names given to classes, variables, methods, etc.	Student, age, calculateSum
Literals	Constant values	100, 3.14, 'A', "Java", true
Operators	Perform operations on operands	+, -, *, /, =, &&, ==
Separators	Symbols that separate program elements	{}, (), [], ;, ,, .

Q9. Data types in JAVA:

There are two data types available in Java

1. Primitive Data Types
2. Non Primitive Types



Primitive Data Types

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

byte

6. Byte data type is an 8-bit signed two's complement integer
7. Minimum value is -128 (-2^7)

8. Maximum value is 127 (inclusive)(2^7 -1)
9. Default value is 0

- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

10. Short data type is a 16-bit signed two's complement integer
11. Minimum value is -32,768 (-2^15)
12. Maximum value is 32,767 (inclusive) (2^15 -1)

- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point

- This data type is generally used as the default data type for real values
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

Summary Table of datatypes in Java:

Data Type	Size	Default Value*	Range	Example
byte	1 byte (8 bits)	0	-128 to 127	byte age = 20;
short	2 bytes (16 bits)	0	-32,768 to 32,767	short marks = 500;
int	4 bytes (32 bits)	0	-2^{31} to $2^{31}-1$	int salary = 50000;
long	8 bytes (64 bits)	0L	-2^{63} to $2^{63}-1$	long population = 8000000000L;
float	4 bytes (32 bits)	0.0f	Approximately ±3.4 × 10^{38} (6–7 digits precision)	float pi = 3.14f;
double	8 bytes (64 bits)	0.0d	Approximately ±1.8 × 10^{308} (15–16 digits precision)	double price = 99.99;
char	2 bytes (16 bits)	'\u0000'	0 to 65,535 (Unicode characters)	char grade = 'A';
boolean	JVM dependent**	false	true or false	boolean result = true;

Q10. Java Statements:

A **statement** in Java is a complete instruction that tells the Java Virtual Machine (JVM) to perform a specific task. Every executable statement usually ends with a **semicolon (;)**.

Example:

```
int x = 10;
```

The above statement declares a variable and assigns a value to it.

Types of Java Statements

Java statements can be broadly classified into the following categories:

1. Declaration Statements
2. Expression Statements
3. Control Statements
4. Block Statements
5. Empty Statement

1. Declaration Statements

These statements are used to declare variables, objects, methods, or classes.

Syntax

```
dataType    variableName;
```

Example

```
int age;
double salary;
String name;
```

2. Expression Statements

An expression statement performs a computation or operation.

Examples

```
int x = 10;
x++;
x = x + 5;
System.out.println(x);
```

These statements perform assignments, increments, method calls, etc.

3. Control Statements

Control statements determine the flow of program execution.

They are classified into three types:

(a) Selection (Decision-Making) Statements

These statements choose one block of code from multiple alternatives.

- if
- if-else
- else-if
- switch

Example:

```
int marks = 80;

if (marks >= 50) {
    System.out.println("Pass");
} else {
    System.out.println("Fail");
}
```

(b) Iteration (Looping) Statements

These statements repeatedly execute a block of code.

- for
- while
- do-while
- Enhanced for loop (for-each)

Example

```
for (int i = 1; i <= 5; i++) {
   System.out.println(i);
}
```

(c) Jump (Branching) Statements

These statements alter the normal flow of execution.

- break
- continue
- return

Example

```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        break;

    System.out.println(i);
}
```

Output

```
1
2
```

4. Block Statement

A block is a group of zero or more statements enclosed within braces {}.

Example:

```
{
    int a = 10;
    int b = 20;
    System.out.println(a + b);
}
```

The statements inside the braces are treated as a single unit.

5. Empty Statement

An empty statement contains only a semicolon (;) and performs no action.

Example

```
;
```

It is sometimes used intentionally in loops.

while (condition);

Summary Table of Statements in Java:

Statement Type	Purpose	Examples
Declaration Statement	Declares variables, methods, classes, or objects	int x;, String name;
Expression Statement	Performs computation or assignment	x = 10;, x++;, sum();
Selection Statement	Makes decisions based on conditions	if, if-else, switch
Iteration Statement	Repeats a block of code	for, while, do-while, for-each

Statement Type	Purpose	Examples
Jump Statement	Alters the normal flow of execution	break, continue, return
Block Statement	Groups multiple statements into a single unit	{ ... }
Empty Statement	Performs no action	;

Q11. Command Line Arguments in Java

Command-line arguments are the values passed to a Java program from the command prompt at the time of program execution. These arguments are stored as **strings** in the String[] args array of the main () method.

Syntax

public static void main(String[] args)
or

public static void main(String args[])

- String[] args is an array that stores the command-line arguments.
- Each argument is stored as a **String**.
- First argument is stored in args[0], the second in args[1], and so on.

Memory Representation

Suppose the command is:

java Demo Apple Mango Orange

Then the *args* array contains:

Index	0	1	2
Value	“Apple”	“Mango”	“Orange”

- args[0] → "Apple"
- args[1] → "Mango"
- args[2] → "Orange"

Example 1: Display Student Details

```
public class Student {
    public static void main(String[] args) {

        System.out.println("Name: " + args[0]);
        System.out.println("Roll No: " + args[1]);
        System.out.println("Branch: " + args[2]);
    }
}
```

Execution

java Student Ravi 101 CSE

Output

```
Name: Ravi
Roll No: 101
Branch: CSE
```

Example 2: Adding Two Numbers

Since command-line arguments are stored as **Strings**, they must be converted into integers before performing arithmetic operations.

```
public class Addition {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum = a + b;
        System.out.println("Sum = " + sum);
    }
}
```

Compilation

```
javac Addition.java
```

Execution

```
java Addition 25 35
```

Output

```
Sum = 60
```

Q12. User Input in Java:

User input is the process of accepting data from the user during the execution of a Java program. Java provides several ways to read user input, such as using the **Scanner class**, **BufferedReader**, **Console**, and **Command-Line Arguments**.

Method	Package	Suitable For
Scanner	java.util	Beginners, simple input
BufferedReader	java.io	Fast input, competitive programming
Console	java.io	Reading passwords and console input
Command-Line Arguments	No package required	Passing input while running the program

1. Using Scanner Class (Most Common Method)

The Scanner class is mainly used to get the user input, and it belongs to the java.util package. In order to use the Scanner class, first import the Scanner

Class from util package, and create an object of the class and use any of the Scanner class methods on Scanner Object.

```
import java.util.Scanner;// Import the Scanner class
Scanner sc = new Scanner(System.in) ;// Create a Scanner object
String name = sc.nextLine() ;// Read user input
```

Scanner Class Methods

There are various methods of Scanner class which can be used for various data types.

Method	Description
nextBoolean()	Reads a Boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads an int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user
next().charAt(0)	Reads a character

Example1:

```
import java.util.Scanner;
public class ScannerDemo1 {
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        String name = sc.nextLine();
        char gender = sc.next().charAt(0);
        int age = sc.nextInt();
        long mobileNo = sc.nextLong();
        double cgpa = sc.nextDouble();


        System.out.println("Name: " + name);
```

```
        System.out.println("Gender: " + gender);
        System.out.println("Age: " + age);
        System.out.println("Mobile Number: " + mobileNo);
        System.out.println("CGPA: " + cgpa);
    } }
```

Input:

Geek

F

40

9876543210

9.9

Output:

Name: Geek

Gender: F

Age: 40

Mobile Number: 9876543210

CGPA: 9.9

Sometimes, we have to check if the next value we read is of a certain type or if the input has ended (EOF marker encountered).

Then, we check if the scanner's input is of the type we want with the help of hasNextXYZ() functions where XYZ is the type we are interested in. The function returns true if the scanner has a token of that type, otherwise false.

For example, in the below code, to check for integer we have used hasNextInt(). To check for a string, we use hasNextLine(). Similarly, to check for a single character, we use hasNext().charAt(0)

Example 2:

```
import java.util.Scanner;


public class ScannerDemo2 {
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int sum = 0, count = 0;


        while (sc.hasNextInt()) {
            int num = sc.nextInt();
            sum += num;
            count++;
        }
        if (count > 0) {
            int mean = sum / count;
            System.out.println("Mean: " + mean);
        }
        else {
                System.out.println("Mean cannot be calculated.");
        }    } }
```

Input

1 2 3 4 5

Output
Mean: 3

2. Using BufferedReader

BufferedReader reads text efficiently from the keyboard.

```
import java.io.*;

public class BufferedReaderDemo {
    public static void main(String[] args) throws IOException {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Enter your name: ");
        String name = br.readLine();

        System.out.println("Name: " + name);
    }
}
```

Advantages

- Faster than Scanner.
- Suitable for large input.

Disadvantages

- Reads input as String.
- Requires exception handling (IOException).

3. Using Console Class

The Console class is available in java.io and is mainly used for secure input, such as passwords.

```
import java.io.Console;

public class ConsoleDemo {
    public static void main(String[] args) {

        Console con = System.console();

        String name = con.readLine("Enter Name: ");

        System.out.println("Welcome " + name);
    }
}
```

4. Using Command-Line Arguments

Input is provided while running the program.

```
public class CommandDemo {
    public static void main(String[] args) {

        System.out.println("Name : " + args[0]);
        System.out.println("Age  : " + args[1]);
    }
}
```

Compile

```
javac CommandDemo.java
```

Execute

```
java CommandDemo Ravi 20
```

Output

```
Name : Ravi
Age  : 20
```

Q13. Escape Sequences in Java

Escape Sequences in Java are special character combinations that begin with a backslash (\). They are used to represent characters that are difficult or impossible to type directly in a string or character literal.

Escape Sequence	Description	Example	Output
\n	New line	"Hello\nWorld"	Hello World
\t	Horizontal tab	"Java\tProgramming"	Java Programming
\b	Backspace	"ABC\bD"	ABD
\r	Carriage return	"Hello\rJava"	H e l l o J a v a ↓ ↓ ↓ ↓ J a v a o
\f	Form feed	"Hello\fWorld"	Hello Page break (rarely used) World
\'	Single quote	'\''	'
\"	Double quote	"She said, \"Hi\""	She said, "Hi"
\\	Backslash	"C:\\Users\\Admin"	C:\Users\Admin

Q14. Comments in Java

Comments are non-executable statements used to explain the code, improve readability, and make maintenance easier. The Java compiler ignores comments during compilation.

Types of Comments in Java

1. Single-Line Comment (//)

- Used for a single line of explanation.
- Begins with //

Syntax:

```
// This is a single-line comment
```

Example:

```
public class SingleLineComment {
    public static void main(String[] args) {
        // Displaying a message
        System.out.println("Welcome to Java");
    }
}
```

Output:

```
Welcome to Java
```

---

2. Multi-Line Comment (/* ... */)

- Used for comments spanning multiple lines.
- Begins with /* and ends with */

Syntax:

```
/*
   This is a
   multi-line comment.
*/
```

Example:

```
public class MultiLineComment {
    public static void main(String[] args) {

        /*
         This program prints
```

```
        a welcome message.
       */

       System.out.println("Hello Java");
    }
}
```

Output:

Hello Java

---

3. Documentation Comment (/** ... */)

- Used to generate API documentation using the Javadoc tool.
- Begins with /** and ends with */.
- Commonly placed before classes, methods, constructors, and fields.

Syntax:

```
/**
 * Documentation comment
 */
```

Example:

```
public class Calculator {

    /**
     * Adds two integers.
     *
     * @param a First number
     * @param b Second number
     * @return Sum of a and b
     */
    public int add(int a, int b) {
        return a + b;
    }
}
```

Comparison of Java Comments

Type	Symbol	Purpose
Single-line	//	Explains a single statement or line
Multi-line	/* ... */	Explains multiple lines or temporarily disables code
Documentation (Javadoc)	/** ... */	Generates API documentation using Javadoc

---

Advantages of Comments

- Improve code readability.
- Make programs easier to understand and maintain.
- Help explain complex logic.
- Useful for debugging by temporarily commenting out code.
- Documentation comments help generate professional API documentation.

Summary

Comment Type	Syntax	Usage
Single-line	//	Short explanations for one line
Multi-line	/* ... */	Long explanations or temporarily disabling code
Documentation	/** ... */	API documentation generation using Javadoc