CORE JAVA - COURSE

Preface:

Welcome to the world of Java programming! Whether you are a student, a professional developer, or an enthusiast looking to enhance your programming skills, this book is designed to guide you through the core concepts of Java. A language that has revolutionized software development since its inception. This book aims to equip you with a strong foundation in Java, ensuring that you are well-prepared to tackle a wide range of programming challenges. The objective of this course is to teach the learner how to use Object Oriented paradigm to develop code and understand the concepts of Core Java and to cover-up with the pre-requisites of Core java

Book Description:

Designed to support an introductory programming course, this book teaches you concepts of problem-solving and object-orientated programming using a fundamentals-first approach. As beginner programmers, you learn critical problem-solving techniques then move on to grasp the key concepts of Java programming. The book transitions smoothly through a carefully selected set of procedural programming fundamentals to object-oriented fundamentals.

Purpose of This Book:

The primary goal of this book is to provide a comprehensive introduction to the core features of Java, focusing on the concepts, techniques, and tools that are essential for mastering the language. We start with the basics, covering the syntax and semantics of Java, before delving into more advanced topics such as object-oriented programming, data structures, exception handling, and multithreading.

How This Book is Structured:

The book is organized into several chapters, each focusing on a specific aspect of Java programming. Early chapters introduce fundamental concepts such as variables, data types, and control structures, while later chapters explore more complex topics like object-oriented design, file handling, and networking. Each chapter includes code examples and exercises designed to reinforce the material and encourage you to apply what you've learned.

Advantages:

- The best Java tutorial from scratch, for completing a book written in a living language.
- Explanations with real life examples.
- Traditional thoughtful presentation of the basics.
- Good presentation of topics on collections and generics.
- Short and good presentation.
- It seems to have all the topics that a beginner needs to know, and not just a beginner.

Required Software:

To compile and run all of the programs in this book, you will need the latest Java Development Kit (JDK) from Oracle, which, at the time of this writing, is JDK 22. This is the JDK for Java SE 22. Instructions for obtaining the Java JDK are given in Chapters. If you are using an earlier version of Java, you will still be able to use this book, but you won't be able to compile and run the programs that use Java's newer features.

About the Author:

Krishna N. is an IT trainer with over 12+ years of experience in both training and real-time industry. He holds postgraduate degrees in MCA (Master of Computer Applications) and MTech from the Department of Computer Science.

CORE JAVA - COURSE INDEX PAGE

Core Java

Introduction (History) to Java:

Java is a popular programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by **Sun Microsystems** (which is now the subsidiary of Oracle) in the year 1995. Work on the language began in 1991, the programming language was released under the name "Oak". "**James Gosling** is known as the father of Java".

Initially, the new language was called "**Oak**". Under the window of James Gosling's office, there was an oak tree which he liked a lot, and so he decided to name the new language in honour of the tree. When Sun Microsystems saw that the language worked well and could be made public, they decided to change the name, since another programming language already had that name. Also, they needed a better-sounding name, one that was more commercial.

So, they called it "Java." May 23, 1995 – the official release date under the "Java" name. In 2009, the Java programming language got a new owner when Sun Microsystems was acquired by **Oracle**. Currently, Oracle owns Java. According to data from Oracle, more than 3 billion devices in the world run on Java.

Platform:

Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform. According to Sun Microsystems, 3 billion devices run Java. There are many devices where Java is currently used. **Some of them are as follows:**

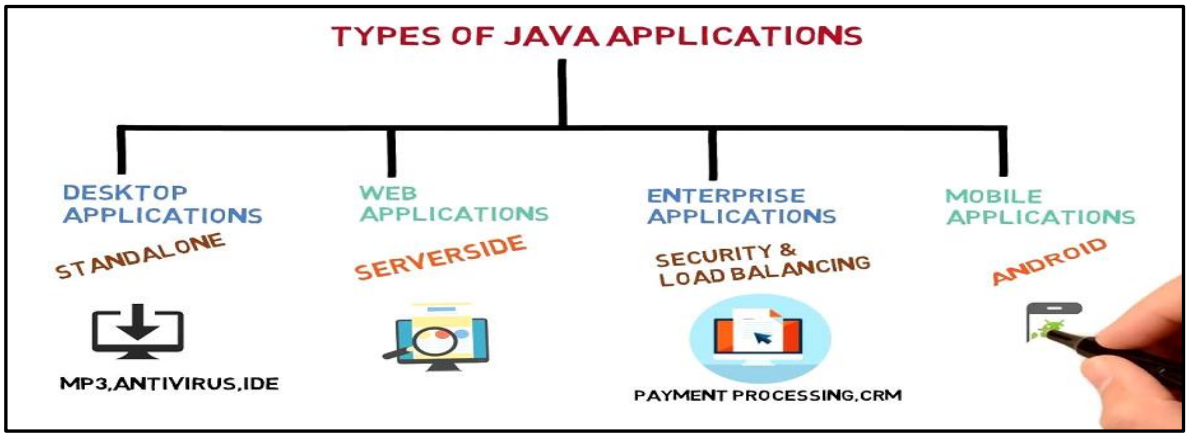
- Desktop Applications such as **Acrobat Reader, Media Player, Antivirus, etc**.
- Web Applications such as **irctc.co.in**, **google.com, etc.**
- Enterprise Applications such as **Banking Applications**.
- Mobile Applications such as **Netflix, Spotify, Twitter, WhatsApp.**
- Embedded System such as **Electronics of cars, Telephones, Modems, Robots and Digital Watches.**

Types of Java Applications:

There are mainly 4 types of applications that can be created using Java programming:

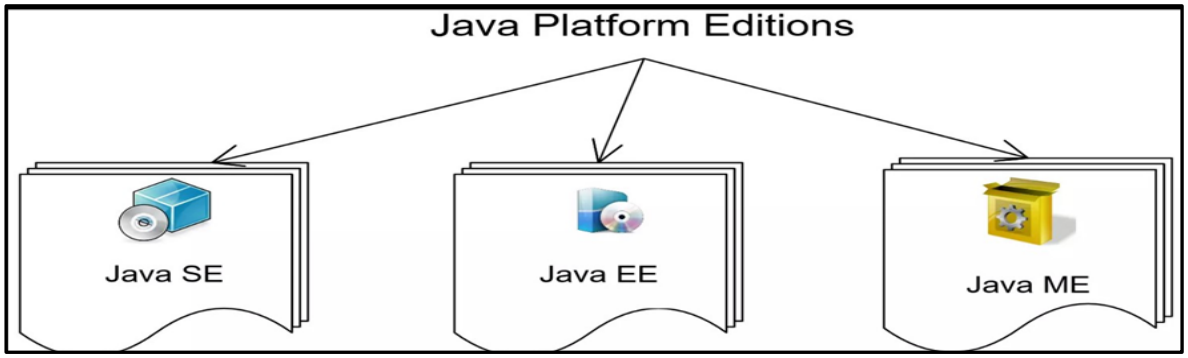
1. **Standalone Application:** Standalone applications are also known as **Desktop Applications** or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.
2. **Web Application:** An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3. **Enterprise Application:** An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.
4. **Mobile Application:** An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.



Java Platforms or Editions:

1. **Java SE (Java Standard Edition):** It is a Java programming platform. It is also called as Java 2 Standard Edition (abbreviated as J2SE). It is used to develop simple Java applications. With this Java edition, you can create console applications, applets, and user interface applications. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.
2. **Java EE (Java Enterprise Edition):** The Java Enterprise Edition or the Java 2 Enterprise Edition (abbreviated as Java EE or J2EE) is used by applications for large corporations to develop applications for banks, insurance companies, retailers, etc. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.
3. **Java ME (Java Micro Edition):** The Java Micro Edition or the Java 2 Micro Edition (abbreviated as Java ME or J2ME) is used to develop applications for cell phones, PDA's, and other low-power computer systems. It is a micro platform that is dedicated to mobile applications.



For Java Developer:

Programmers who work with JAVA PROGRAMMING LANGUAGE is great demand in worldwide. Java programmers who work for medium and large companies, such as banks, insurance companies, large retailers, etc. To create an application for such companies, you need to know not only the Java language, but also specifics of the industry you are working for. Therefore, every time a Java developer starts working at a new company, the programmer has to learn the specifics of the new industry. It's the only way to create a good application. Suppose you want to create a bank application on derivatives(features). First, you will need to become an expert in derivatives. Otherwise, how can you build such an application? Or, say you

are wanting to work on an application for a large retail company. The only way to do it is to first learn the specifics of the retail business.

For this reason, Java programmers belong to the programming best of the best. So, if you like learning a lot in a short period of time and under strict deadlines, then the life of a Java programmer may be the one for you.

It's no surprise that those who've just started learning their first programming language always have a lot of questions. But trust me, if you simply have patience and study Java every day, your understanding will quickly grow.

Java Version History:

Many java versions have been released till now. The current stable release of Java is Java SE is 8 and 11.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014) [**Stable Version**]
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018) [**Stable Version**]
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (March 2022)
21. Java SE 19 (September 2022)
22. Java SE 20 (March 2023)
23. Java SE 21 (September 2023)
24. Java SE 22 [Current Version]
25. Java SE 23 (To be released September 2024)

Note: Since Java SE 8 release, the Oracle corporation follows a pattern in which every **EVEN** version is release in March month and an **ODD** version released in September month.

Where is Java used:

1. **Java is a language used for server applications:** At large corporations. Java is often used in banks, insurance companies, retail networks, etc. One more example: the back end of Google+ is written in Java.
2. **Web applications:** Java is commonly used in e-commerce and web applications. For example, if we're talking about the European and American markets, many web applications of government institutions, educational institutions, insurance companies, and the military are written in Java.
3. **Trading applications:** Trading software facilitates the trading and analysis of financial products, such as stocks, options, futures, or currencies. such as Interactive Brokers, E*TRADE, TradeStation.
4. **Android applications:** If you have a phone on Android, then all the applications in it were written in Java using Google and Android API.
5. **Desktop applications and software and development tools:** such as Eclipse IDE, Netbeans IDE, jEdit (Programmer's Text Editor), jDownloader (open-source download management tool.
6. **Embedded Systems:** Java is used on Smart Cards and Sensors. For example, bank cards work on Java.

- Java works on different platforms (Windows, Mac, Linux, 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 a 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.

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords. The syntax of java is easy and convenient to understand. Java provides numerous advantages over other programming languages. Java is based on classes and is object-oriented in nature. It is a high-level language.

Features of Java:

A list of the most important features of the Java language is given below.

1. **Simple:** No need to wite more lines of program and statements also simple.
2. **"Write Once, Run Anywhere" (WORA):** The main advantage of the Java language is that the same code can run on different operating systems: Like Windows, Linux, MacOS. If you were to write using a different programming language, you would have to write 3 different codes: one for Windows, one for Linux, and one for MacOs. This feature ("write once, run anywhere") makes Java cross-platform. You're probably wondering how Java manages this. The diagram below will explain.
3. **Portable:** (Portable TV means we will change one place to another place easily).
   Write program in one system compile into that system and run that program into any system any ware we will get same output.
4. **Object oriented Language:** OOP is a programming paradigm based on the concept of "classes" and "objects." Add class to procedure-oriented language it is converted to objectoriented language. Decreases program complexity (Means reduce length of program).
5. **User-Friendly Syntax:** The creators of Java didn't try to reinvent the wheel. Instead, they: took all the best parts from the C programming language and it's descendant C++ didn't take the parts of C and C++ that they considered unnecessary and not very successful and added innovative touches to the Java programming language and this approach was very successful. Since there is a lot in common between C, C++, and Java, it was easy for programmers to switch from these languages to Java. They didn't have to learn everything from scratch, because many of the constructions of the Java syntax were familiar to them. This helps explain the consequent rapid growth in the popularity of Java among programmers.
6. **Robust:** Robust means very strong.java is robust language because it supports Exception handling and Memory management (Through Garbage Collection-Memory allocation and deallocation automatically->That's why Pointer's concept [completely eliminated].
7. **Multi-Threaded:** Parallelly runs multiple tasks. If we start to run one task or thread but it is stopped in middle java immediately starts next task in other programs if the first task is stooped the remaining tasks also stopes.
8. **Interpreted:** C and C++ supports either Compiler or interpreter but java supports compiler and interpreter compiler used for generating byte code interpreter used for byte code runs in JVM.
9. **High Performance:** Actually, JVM having interpreter but that interpreter is very slow platforms that's why in JVM installed one more compiler that is JIT (Just in time) compiler, so both compiler and interpreter giving high performance.
10. **Distributed:** Java supports many network-based applications and also supports many protocols like TCP/IP.
11. **Dynamic:** Dynamic means run time. In run time java gives different outputs by introducing Applets Concept.

Note: By using java, we can develop console-based applications and GUI applications and Web applications and Enterprise level applications and network-based applications. Once we learn Java, we can easily move to any other programming languages.

Differences between C and C++ and Java:

C	C++	Java
The C language is based on BCPL (Basic Combined Programming Language).	The C++ language is based on the C language.	The Java programming language is based on both C and C++.
It is a procedural language.	It is an object-oriented programming language.	it is a pure object-oriented programming language.
Dennis Ritchie in 1972	Bjarne Stroustrup in 1979	James Gosling in 1991
The code is executed directly.	The code is executed directly.	The code is executed by the JVM.
It is platform dependent.	It is platform dependent.	It is platform-independent because of byte code.
It uses a compiler only to translate the code into machine language.	It also uses a compiler only to translate Java uses the code into machine language.	Java uses both compiler and interpreter and it is also known as an interpreted language.
It generates the .exe files.	It generates .exe file.	It generates .class file.
The source file has a .c extension.	The source file has a .cpp extension.	The source file has a .java extension.
It supports pointer.	It also supports pointer.	Java does not support the pointer concept because of security.
It uses pre-processor directives such as #include, #define, etc.	It uses pre-processor directives such as #include, #define, #header, etc.	It does not use directives but uses packages.
It does not support exception handling.	It supports exception handling.	It also supports exception handling.
It is widely used to develop drivers and operating systems.	It is widely used for system programming.	It is used to develop web applications,mobile applications, and windows applications.
It uses the calloc(), malloc(), free(), and realloc() methods to manage the memory.	It uses new and delete operator to manage the memory.	It uses a garbage collector to manage the memory.

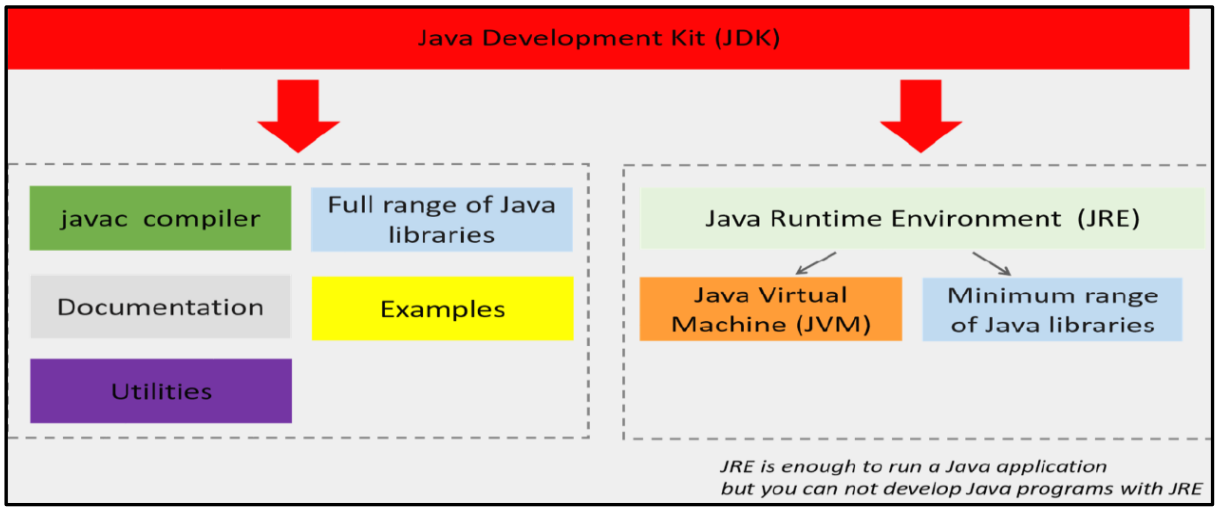
Java Software Configuration (Installation):

In order to begin writing a program in Java, you first need to configure your software environment. And after that you will be able to write your first Java program. To do this, you need to download and install on your computer: **JDK and IDE.**

JDK: (The Java Development Kit) is a kit for Java applications developers. To better understand what JDK consists of, let's look at the diagram below.

JDK consists of:

- Java Runtime Environment (JRE)
- interpreter/loader (Java)
- compiler (javac)
- archiver (.jar)
- documentation generator (Javadoc)
- Full range of Java libraries



Some PCs might have Java already installed. To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

```
C:\Users\HP>java --version
```

If Java is installed, you will see something like this (depending on version):

```
C:\Users\HP>java --version
java 21.0.2 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

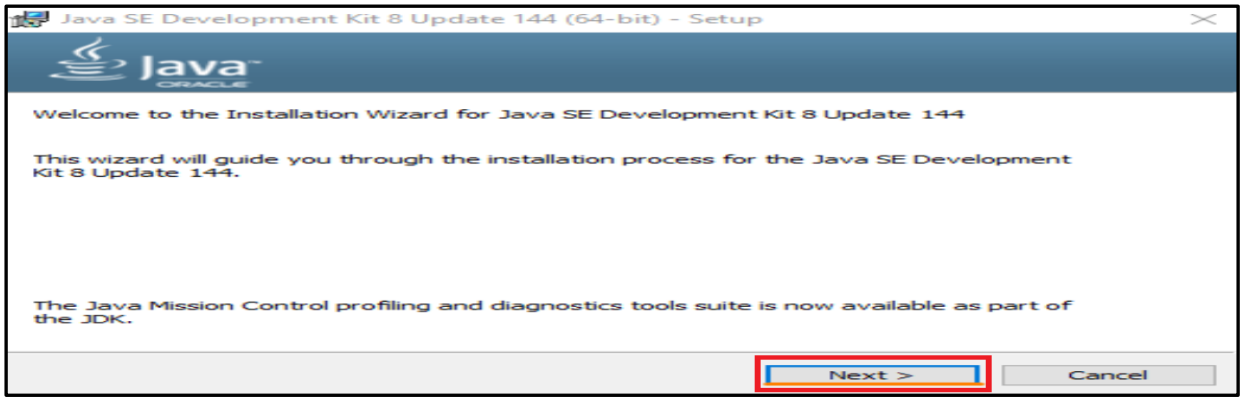
If you do not have Java installed on your computer, you can download any JDK available version free [Open Source].

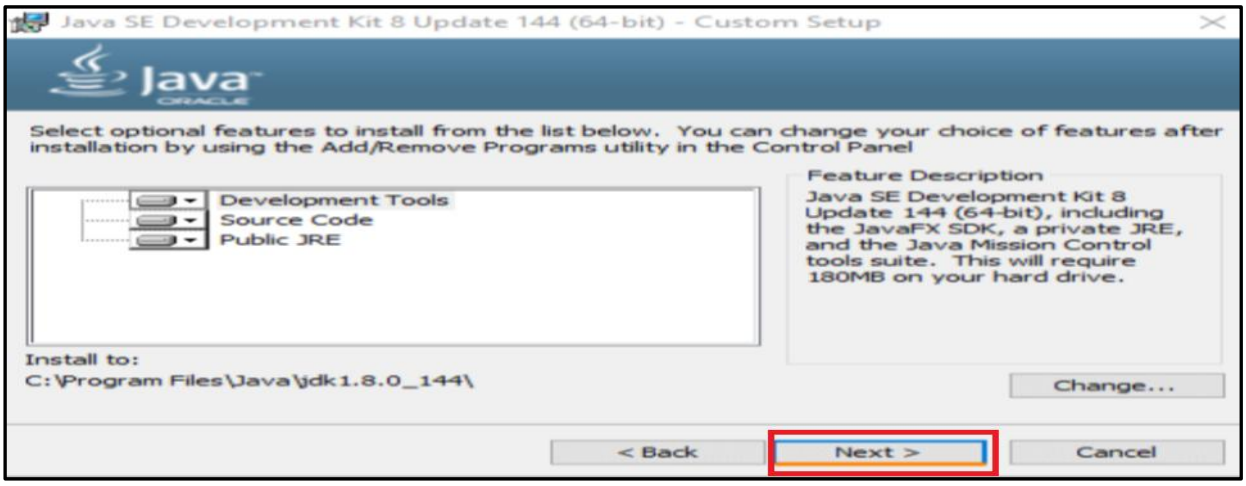
How to Download:

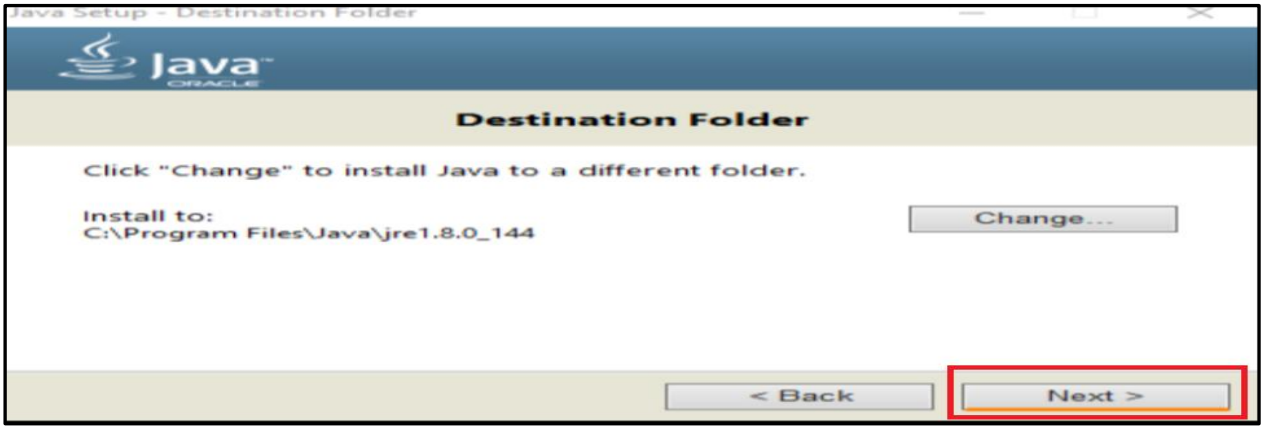
- Search in google download JDK 8 for windows or visit the Oracle website
- Open Official Oracle website.
- jdk-8u401-windows-x64.exe --> .exe file (**Not Zip File**)
- A pop will appear check the box
- Click on download
- A window will open for login but click on create account
- Provide all details and register, give valid email id
- After creating account verify your email-id, confirmation email from oracle

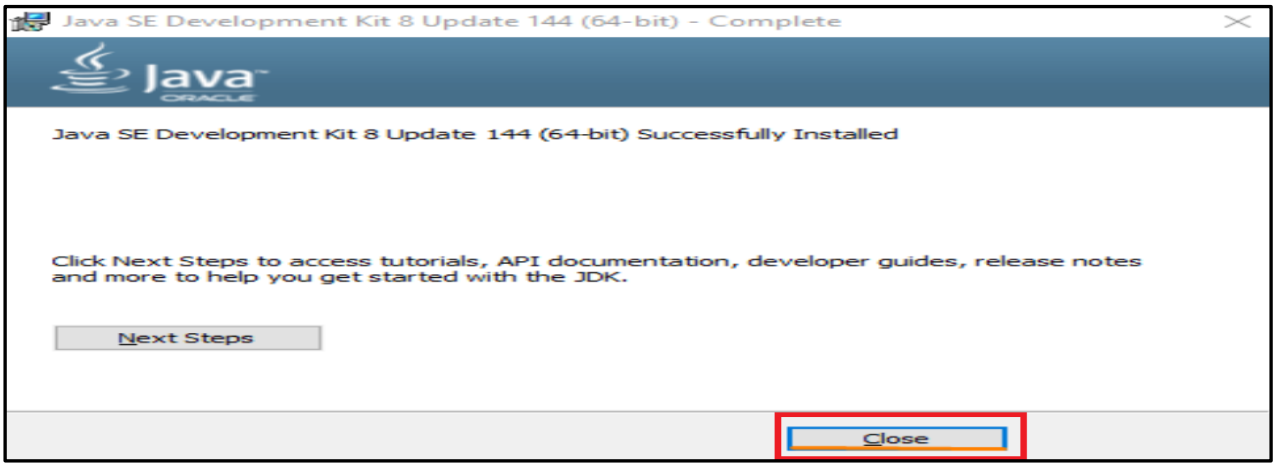
- After verify email enter username and password and sign in in oracle account.
- It will download JDK -11 version
- After downloading double click on .exe file and install. And now it's just the usual process of installing a program.

And now it's just the usual process of installing a program.







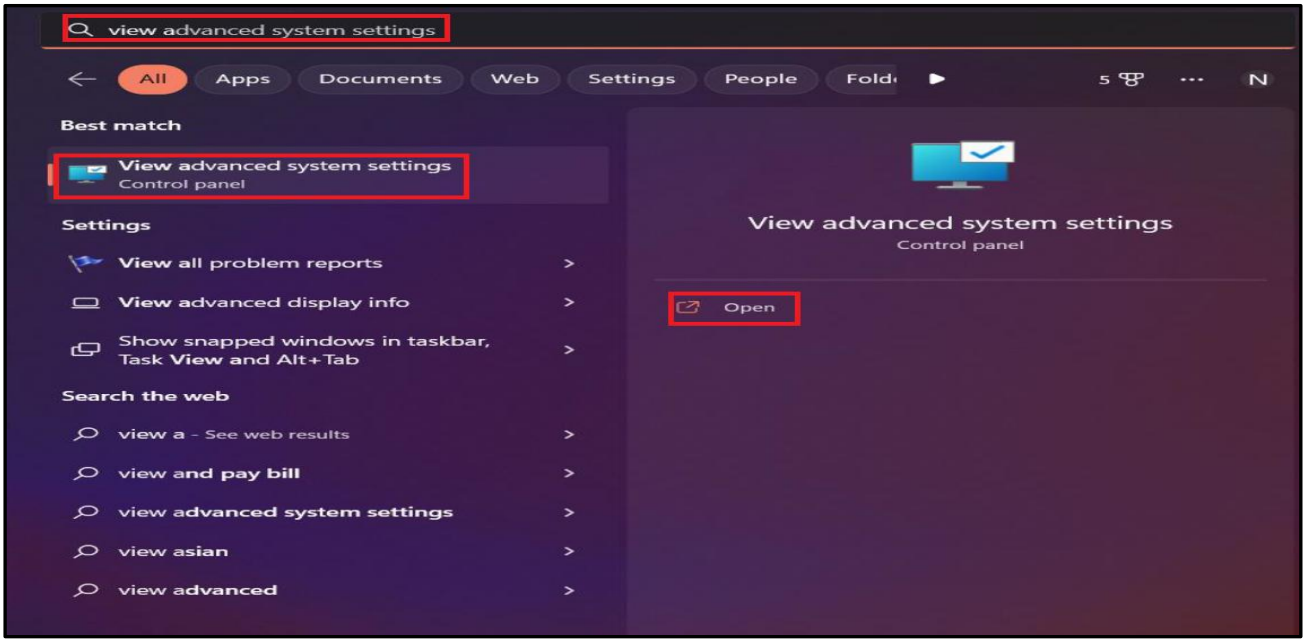


- Confirm in control panel and c-drive Program Files will consist of java folder.

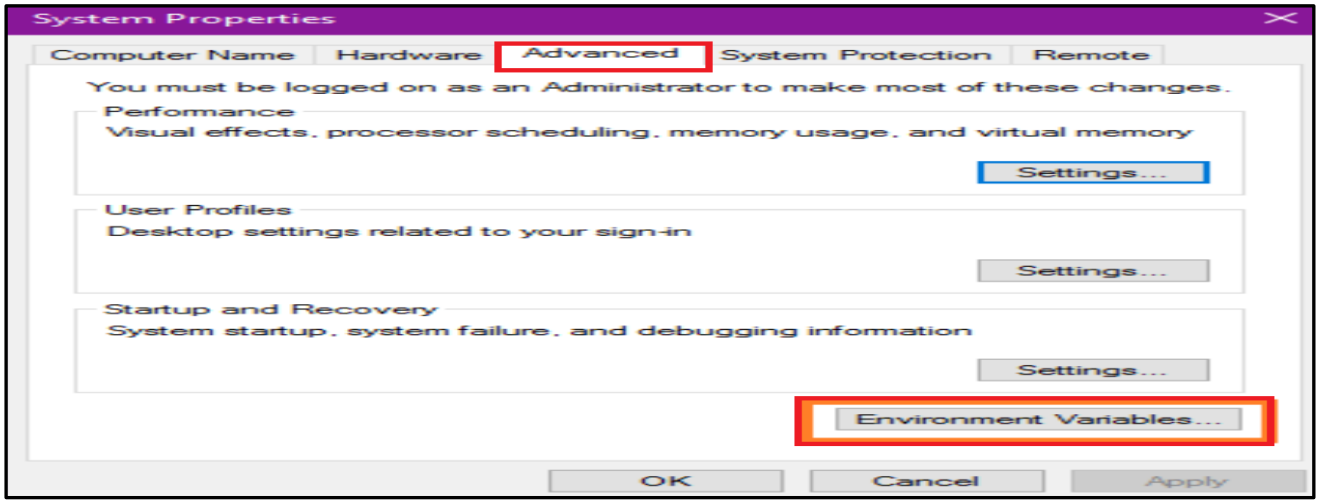
Path Setting: [Java Environment Setup on Windows]

At this point Java is installed, but you still need to set up Path. After install java (JDK) on Windows:

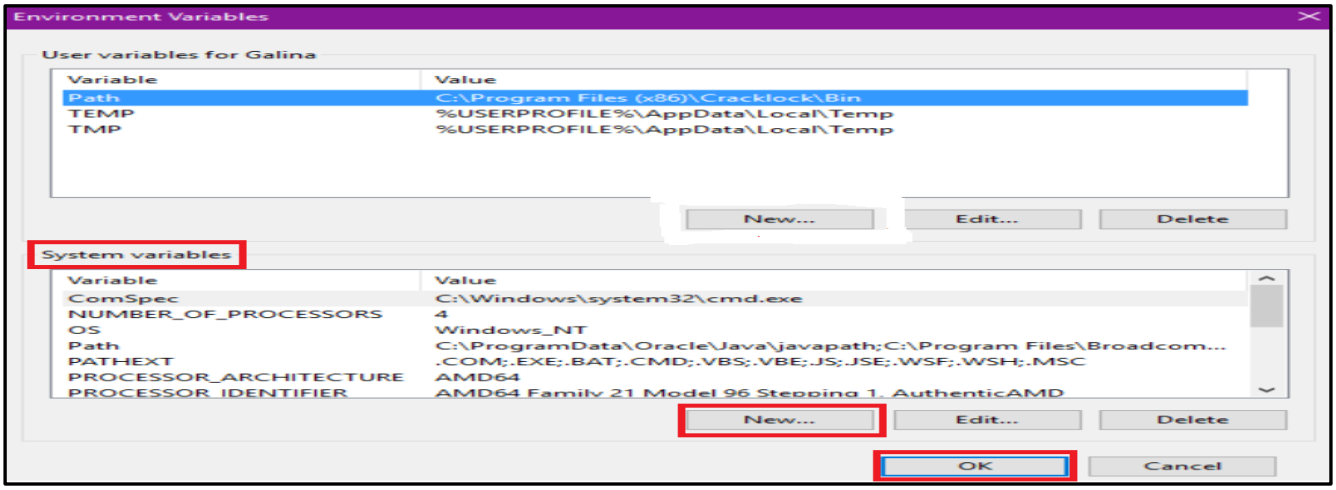
Step1: Search for view Advanced System Settings in pc.



Step2: Click on the "Environment Variables" button under the "Advanced" tab.

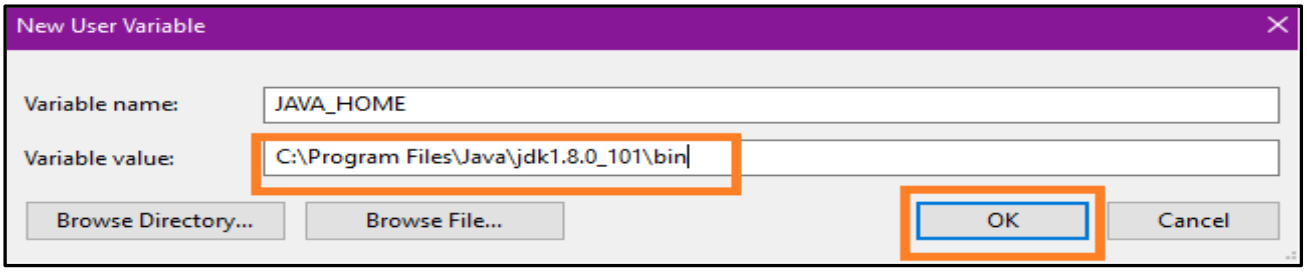


Step3: Inside System Variables Section Click on the "New" button.



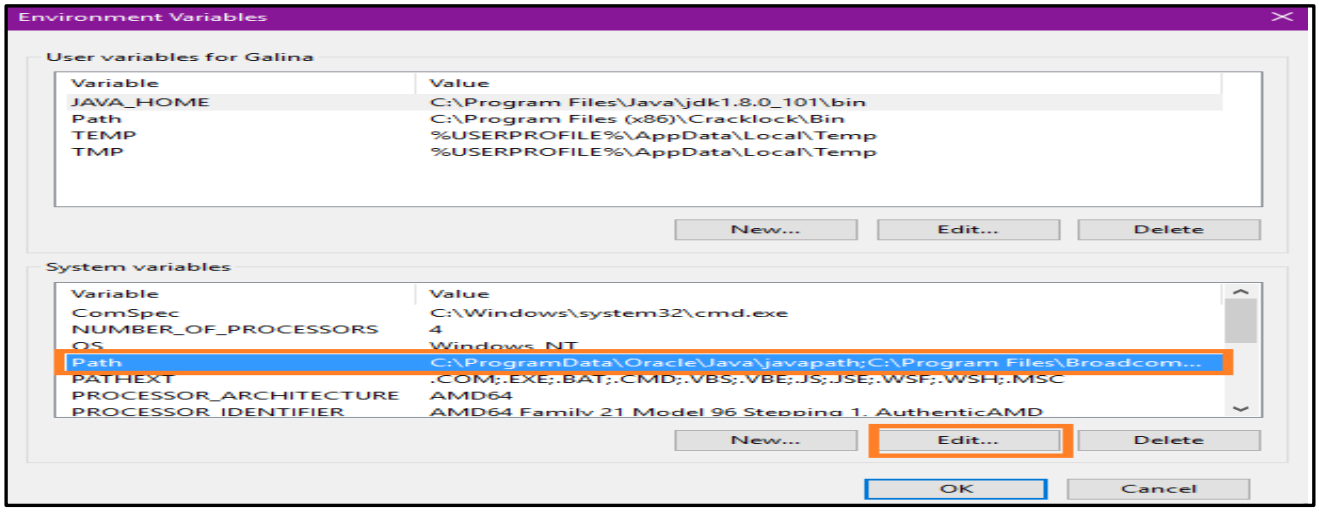
Step4: In the “Variable Name” section, write “JAVA_HOME” or “JAVA_PATH. In the “Variable Value” section, write the path to JDK (i.e. you have to specify where JDK is installed on your computer) or paste the copied path.

For Copy path: Go to **C**-drive--> Program Files --> java--> jdk --> bin-->copy the path [C:\Program Files\Java\jdk-22\bin].



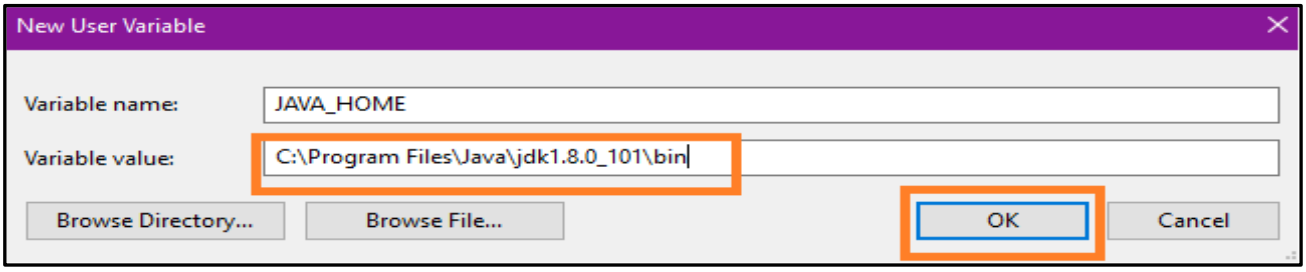
If you want to change existing java path, then follow below steps:

Step1: Now select “Path” in “System variables” and click “Edit” Button.



Step2: In the "Variable Value" section, paste the copied JDK path (i.e. you have to specify where JDK is installed on your computer).

And that's it. We have just configured JDK on Windows together!



- At last, open **Command Prompt** (cmd.exe) and type **java --version** to see if Java is running on your machine. Write the following in the command line (cmd.exe):

```
C:\Users\HP>java --version
```

If Java was successfully installed, you will see something like this (depending on version):

```
C:\Users\HP>java --version
java 21.0.2 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

First Java Program:

To create a simple Java program, you need to create a class that contains the main method.

Write a simple java program to print Hello Java:

To write the java program, you need to open **NotePad** by Start Menu -> All Programs -> Accessories -> Notepad and write a simple program and save it as **Filename.java.** [Ex: **Simple.java**]

To compile and run the above program, you need to open the command prompt by start menu -> **All Programs -> Accessories -> command prompt and go to your current directory first.**

Program:

```
class Simple
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

To Save the program: Filename.java
Simple.java.
To Compile program: javac filename.java
javac Simple.java
To Execute/Run program: java classname
java Simple

Output:
Hello Java

Note: When we compile Java program using javac tool, the Java compiler converts the source code into byte code.

Explanation: Used Parameters are
class: class keyword is used to declare a class in Java.
public: Public keyword is an access modifier that represents visibility. It means it is visible to all.
Static: Static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The
main (): Main method is executed by the JVM, so it doesn't require creating an object to invoke the main () method. So, it saves memory.
void: Void is the return type of the method. It means it doesn't return any value.
main represents the starting point of the program.
String [] args or String args[]: String[] args is used for command line argument.
System.out.println(): It is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

Class and Objects:
Class:
Class is a template or blue print. Class is logical existence. Class is a collection of objects. class is a collection of variables and methods. A document in which features and functions are defined i.e known as Class. Class is a keyword always write in lowercase only. class follows with classname and no semicolon and follows with body i.e open brace and close brace ({}) inside that will write code.
Syntax:
class Classname
{
code
}
Rules to write class Name:
1. Class name is user defined (means we can give any name but follow below rules).
2. Class name must start with uppercase alphabet, lower case alphabet is acceptable but it is warning, according to naming convention rules of java class name starts with uppercase alphabet.
3. Class name should not start with numbers or special characters.
4. Class name should not contain spaces.
5. Class name as "sample" here "s" is lowercase no errors but it is warning

Example: Valid and Invalid Class Names
classname as **Sample** -→ valid
classname as **9sample**-→ invaild
classname as ***sample** -→invalid
classname as **Sample9**-→valid
classname as **Sample Program**-→ invalid

classname as **Sample_Program**-→ valid
Inside class body we have to write code this code will not run because there is no main method.

Objects:

An object in Java is a basic unit of Object-Oriented Programming and represents real-life entities. Objects are the instances of a class that are created to use the attributes and methods of a class. A typical Java program creates many objects, which as you know, interact by invoking methods. Objects correspond to things found in the real world. Basically, an object is created using the new keyword and the state of an object is stored in a variable, whereas the behavior of an object is displayed using functions (methods).

- A thing which is having identity, state, behaviour known as Object.
- Object is an instance of class.
- Object is physical existence.
- Object inherits features of a class.
- Classes and objects are the two main aspects of object-oriented programming.

The state of an object is stored in fields (variables), while methods (functions) display the object's behavior.

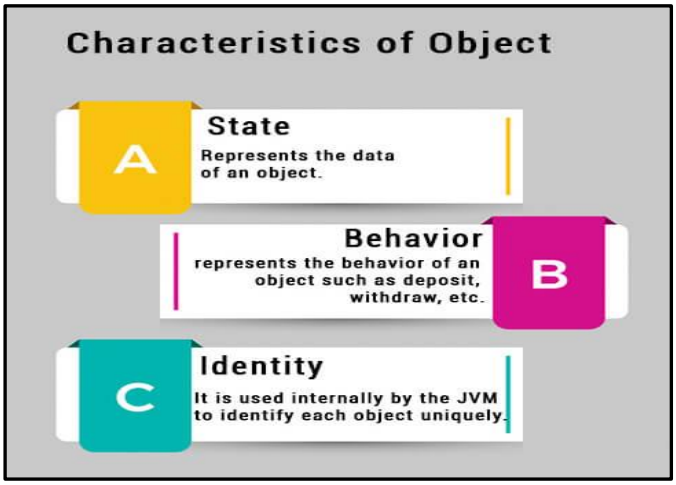
An object has three **Characteristics**:

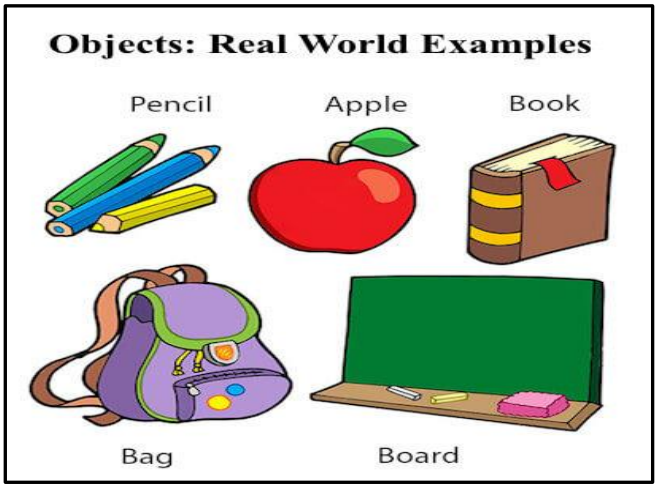
State: Represents the data (value) of an object.

Behaviour: Represents the behaviour (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For **Example**, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.





So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

So basically, an object is created from a class. In Java, the new keyword is used to create new objects. To create an object of Student, specify the class name, followed by the object name, and use the keyword new. When we create an object of the class by using the new keyword, it allocates memory (heap) for the newly created object and also returns the reference of that object to that memory.

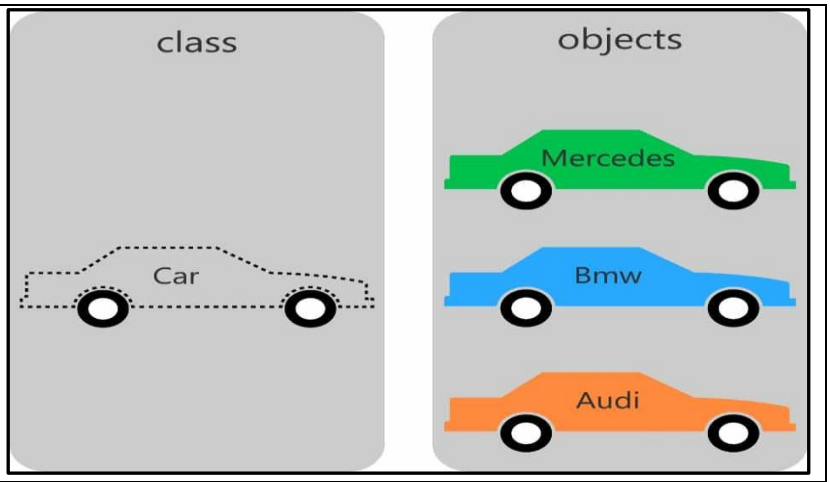
Syntax for creating an object:

ClassName object = new ClassName();

Ex: Student s1=new Student();

Examples of Class and Objects

Class	Object
Human	Man, Woman
Fruit	Apple, Banana, Mango, Guava…
Smart Phone	iPhone, Samsung, Oppo..
Animal	Lion, Elephant, Giraffe
Employee	Krishna, Sujatha, Murali…
Car	BMW, Audi, Mercedes



Difference between Class and Object:

CLASS	OBJECT
Class is a blueprint or template from which objects are created.	Object is an instance of a class.
Class is a group of similar objects.	Object is a real-world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.
Class is a logical entity.	Object is a physical entity.
Class is logical existence	Object is physical existence
Memory space is not allocated, when it is created.	Memory space is allocated, when it is created.
Class is declared using class keyword e.g. class Student {}	Object is created through new keyword mainly e.g. Student s1=new Student ();
Class is declared once.	Object is created many times as per requirement.
Class doesn't allocated memory when it is created.	Object allocates memory when it is created.
The class has to be declared only once.	Objects can be declared several times depending on the requirement.
There is only one way to define class in java using class keyword.	There are many ways to create object in java such as new keyword, new Instance () method, clone () method, factory method and deserialization.

Java Character Set:

Characters are the smallest units (elements) of Java language that are used to write Java tokens. A character set in Java is a set of alphabets, letters, and some special characters that are valid in java programming language. In Java for every character there is a well-defined Unicode code units which is internally handled by JVM. Whenever we write any Java program then it consists of different statements. Each Java Program is set of statements and each statement is set of different Java programming expression. In Java Programming each and every character is considered as a single expression.

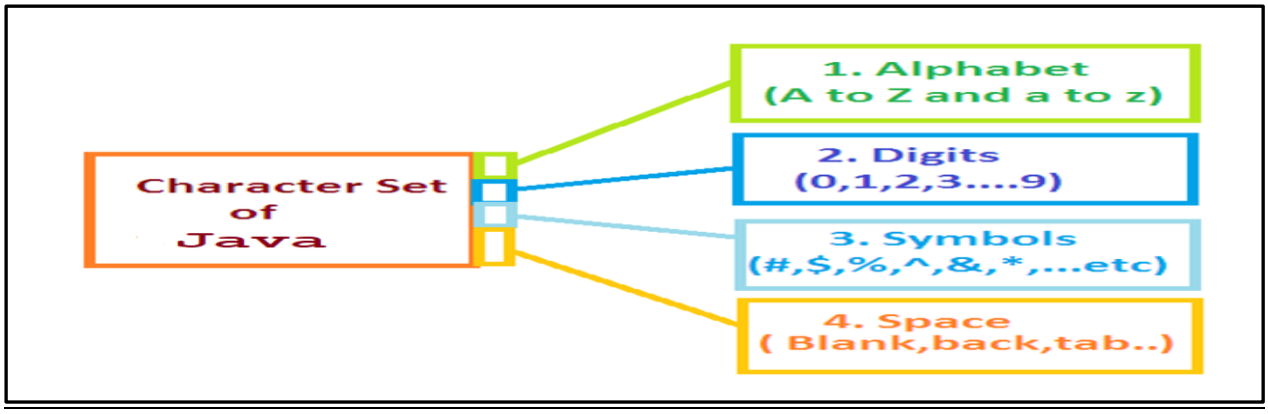
Java language uses the character sets as the building block to form the basic elements such as identifiers, variables, array, etc in the program. These are as follows:

Letters: Both uppercase (A, B, C, D, E, etc.) and lowercase (a, b, c, d, e, etc.) letters.

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Special symbols: _, (,), {, }, [,], +, -, *, /, %, !, &, |, ~, ^, <, =, >, $, #, ?, Comma (,), Dot (.), Colon (:), Semi-colon (;), Single quote (‘), Double quote (“), Back slash (\).

White Space: Space, Tab, New line.



Java Coding Standards: The Java Coding Standard is based on the Java language coding conventions.

- Use descriptive and appropriate names for variables, method names, class names, function names, constants, etc..
- Use meaningful comments every 3-7 lines of code to describe the purpose of classes, methods, and complex algorithms.
- Class names start with an upper-case letter.
- Variable names should be meaningful and written in lower Case (e.g., numberOfStudents). Method names start with a lower-case letter.
- Avoid unnecessary white spaces.
- Curly braces ("{}") are used to define the bodies of classes, methods, and loops.
- Open braces (i.e. "{") do not start a new line.
- Close braces (i.e. "}") do start a new line, and are indented with the code they close.
- Semicolon is a part of syntax in Java. Semicolons are also used to terminate the statements.
- Organize classes into packages that reflect their functionality.
- Always handle exceptions appropriately using try-catch blocks.

JDK, JRE and JVM:

JDK: Java Development Kit (JDK) is a software development kit that develops applications in Java. JDK provides development tools and run time environment. JDK contains JRE along with various development tools like Java libraries, Java source compilers, Java debuggers, bundling and deployment tools. We need the JDK to convert your source code into a format that the Java Runtime Environment (JRE) can execute.

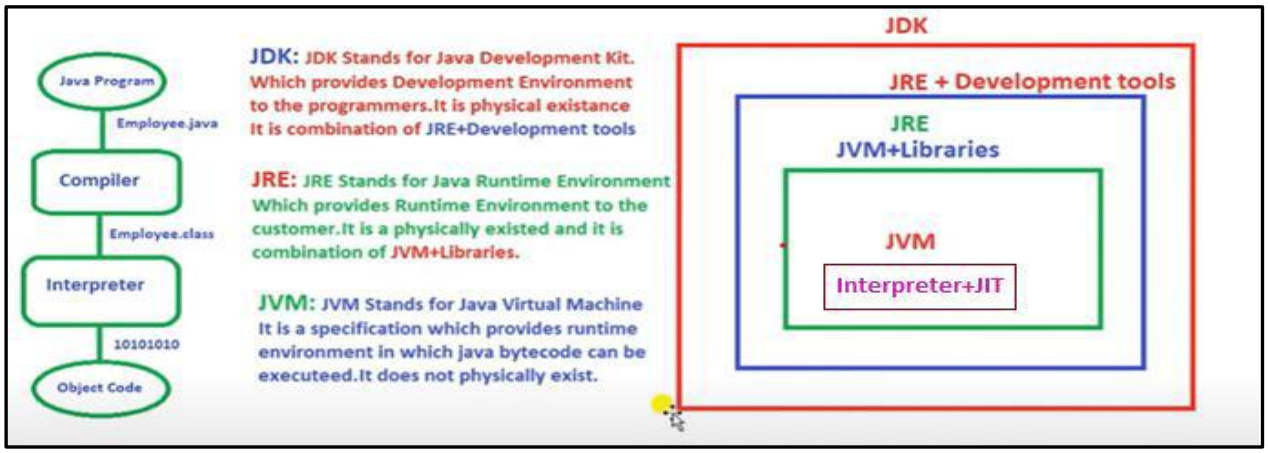
JRE: Java Runtime Environment (JRE) is an implementation of the JVM. The JRE is software that Java programs require to run correctly. The JRE is the underlying technology that communicates between the Java program and the operating system. JRE is part of a Java development kit (JDK). A JRE is made up of a Java virtual machine (JVM), Java class libraries, and the Java class loader. JRE identifies all the helpful class libraries needed for execution. Java developers are initially presented with two JDK tools, java and javac. Both are run from the command prompt. Java source files are simple text files saved with an extension of .java. After writing and saving Java source code, the javac compiler is invoked to create .class files. Once the .class files are created, the 'java' command can be used to run the java program.

JVM: Java Virtual Machine (JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an interpreter. JVM have both compiler and interpreter. Because the compiler compiles the code and generates bytecode. After that the interpreter converts bytecode to machine understandable code.

JVM is a subclass of JRE that decodes the bytecode into machine language and other minor tasks. JVM and JRE do not participate in development processes like debugging and compiling; JDK is used for them.

JIT: Jit stands for just intime compiler. It is responsible to help interpreter for executing java byte code fast. Basically, Jit will improve the performance of java program execution.

Note: Users are required to install a JDK only. If you install JDK then JRE will already be packaged in it and installed automatically along with JDK.JVM is a part of JRE. It cannot be separately downloaded and installed. JDK includes both JVM and JRE.so you do not need to download and install the JRE and JVM separately in your PC.



IDE: IDE is a tool designed for software development. IDE full form is Integrated Development Environment. Java IDE is a software application that enables users to write and debug Java programs more easily. Java IDEs include a code editor, a compiler, a debugger, and an interpreter.

The most popular IDE's:

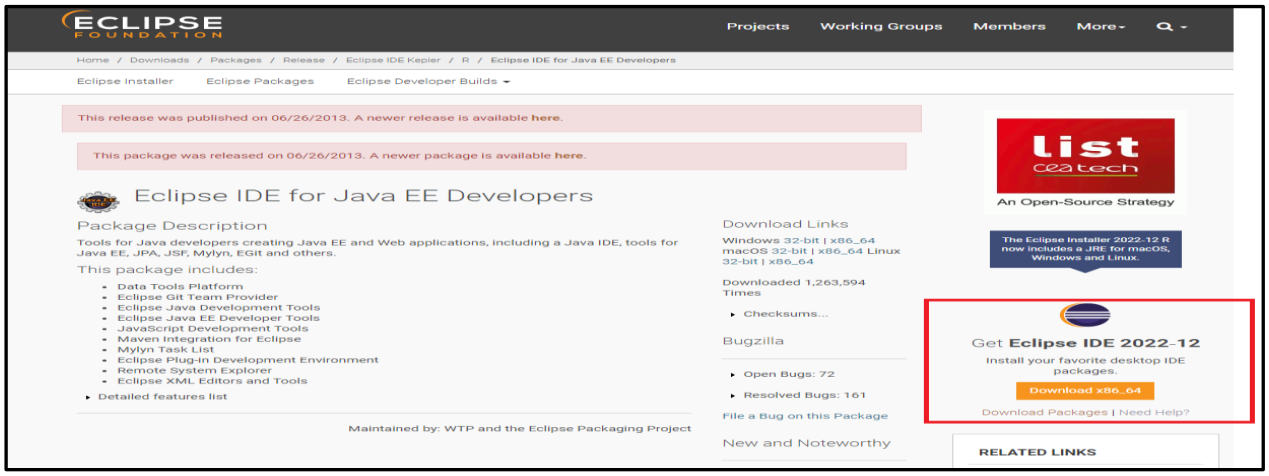
1. Eclipse
2. JetBrains IntelliJ IDEA
3. NetBeans

Many other IDEs are most popular in the Java IDE's that can be used according to our requirements.

Eclipse: Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language. Eclipse is an open-source IDE developed by IBM., which is freely available. Go to its official website, download and install for free.

Download and Installation of Eclipse:

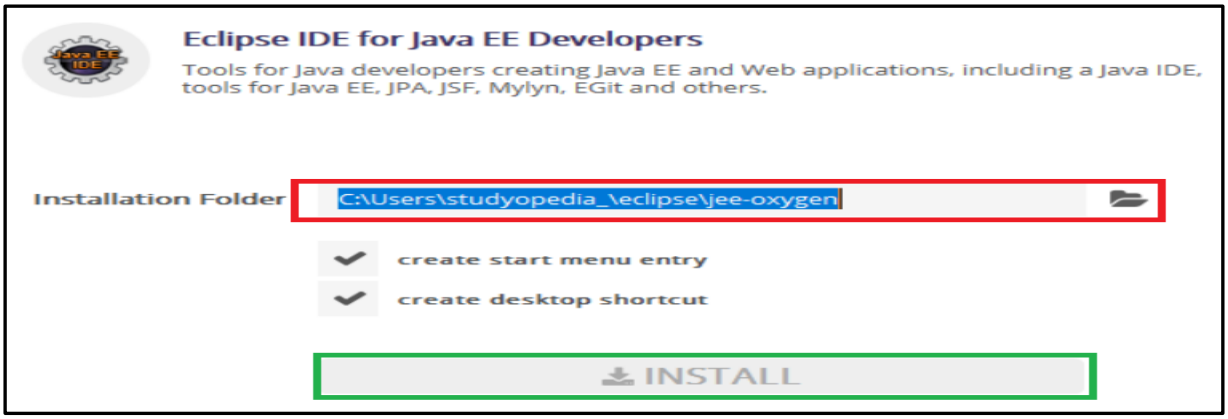
- ✓ Visit this page: https://www.eclipse.org/downloads/packages/release/kepler/r/eclipse-ide-java-ee-developers
- ✓ Under try this eclipse installer click windows x86_64.



- ✓ Click on .exe file.

- ✓ .exe file will be downloaded.
- ✓ After downloading the eclipse-inst-win64.exe package, click on it and begin installing Eclipse IDE,
- ✓ Now, select the Eclipse Installer you want,
- ✓ A popup will be opened in that select Eclipse IDE for Java developer.

- ✓ Click on install.
- ✓ After clicking, you will be given an option to change the Eclipse installer location,

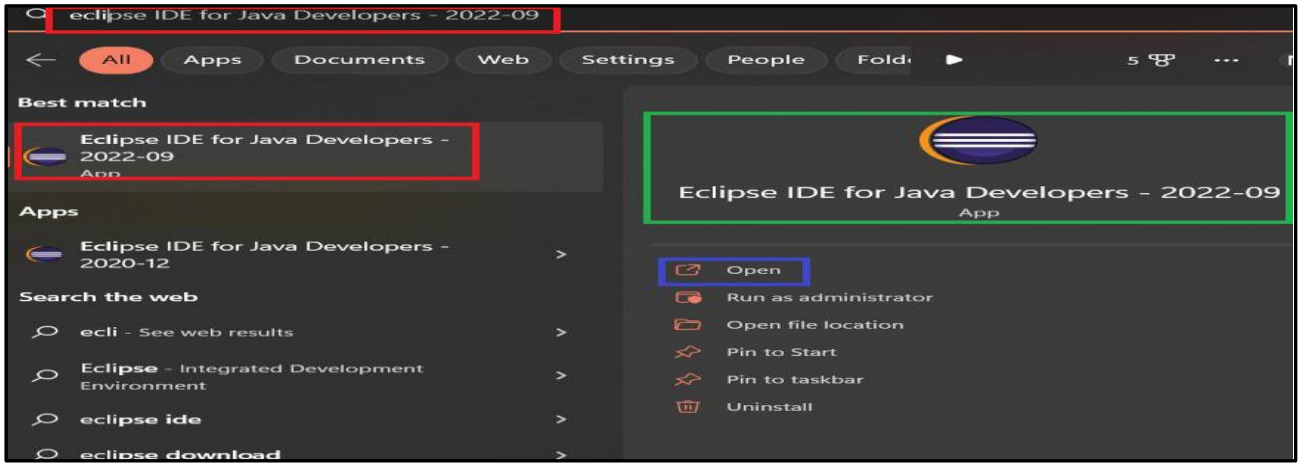


- ✓ Read and Accept the Eclipse Foundation Software User Agreement,

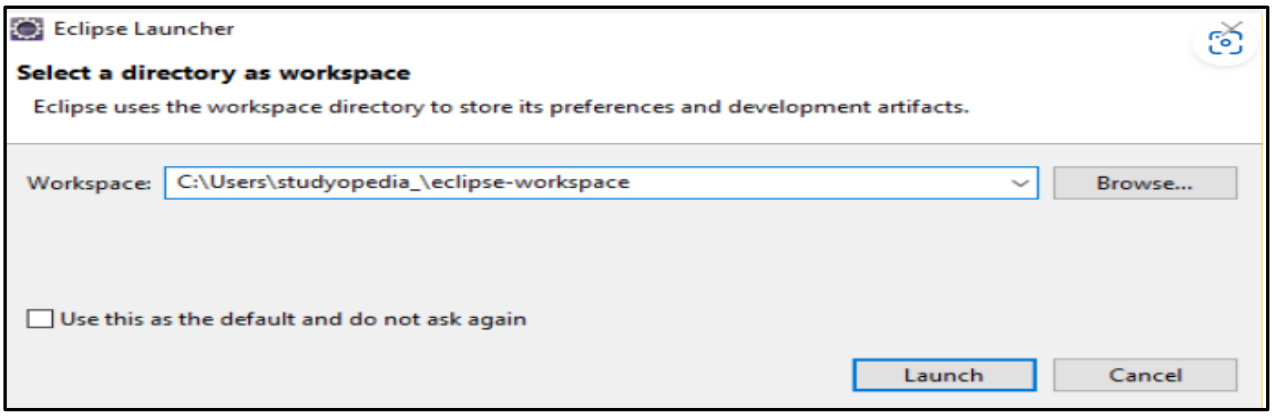
✓ The installation starts…

✓ After some minutes, the installation will complete. Now, click on Launch to open Eclipse IDE for the first time.

✓ The Launch button shown above will only be available once, when you will install Eclipse.
✓ Here, we will see how to start Eclipse and use it whenever you wish to open the Eclipse IDE software,
✓ Go to START, type Eclipse and click on it.



✓ The Eclipse IDE will open and it will ask you to add directory location for Workspace. This is the project location.



✓ After setting the directory above, click Launch.

✓ The Welcome screen is visible now.

✓ Now, begin working on Eclipse IDE. If you want the welcome screen to be visible always, then select Always show Welcome at start up.

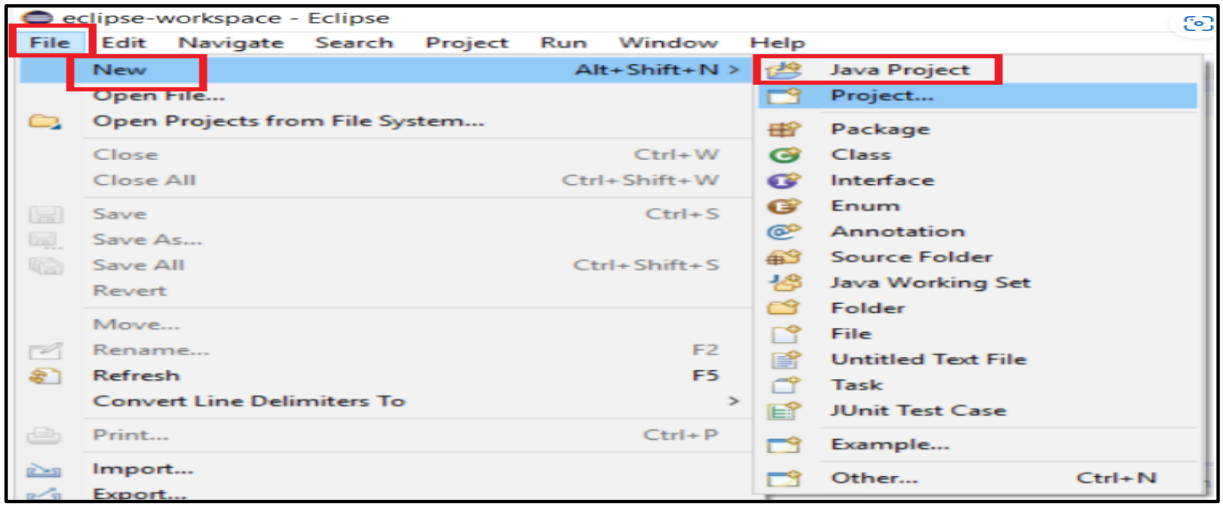
Java Project in Eclipse:

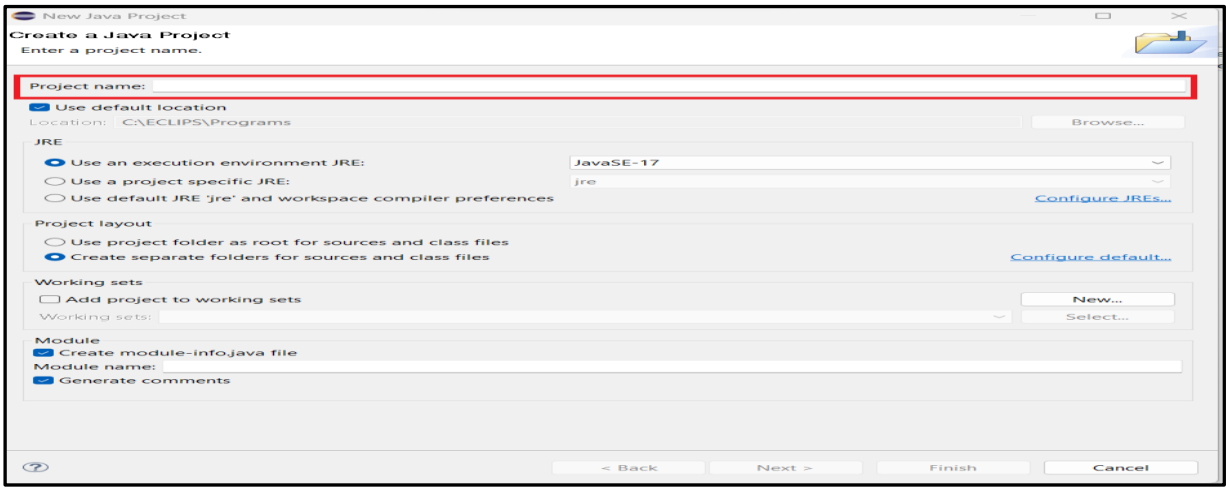
How to create Java Project in Eclipse?

This step will show you how to create a Java application Project in Eclipse.

Step1: Install the Eclipse IDE for Java Developers. When you install Eclipse for the first time, you're given the option to choose your IDE (integrated development environment). Choose "Eclipse IDE for Java Developers". This will install the necessary files and tools to create Java projects.

Step2: Click "File" → "New" → "Java Project". This will open the "New Java Project" window.





Step3: Give the project a name. This doesn't have to be the final name for your program, but should help you and others identify the project.

Step4: Select the location for the project files. The files are saved to the Eclipse directory by default. You can set a custom location if you prefer.

Step5: Select Java Runtime Environment (JRE) you want to use. If you are creating a program for a specific JRE, select it from the drop-down menu. By default, the newest JRE will be selected.

Step6: Click "Finish" to start working on your new project. You'll be taken to your Java workspace.

Note: Your project will be displayed in the "Package Explorer" frame on the left side of the window.

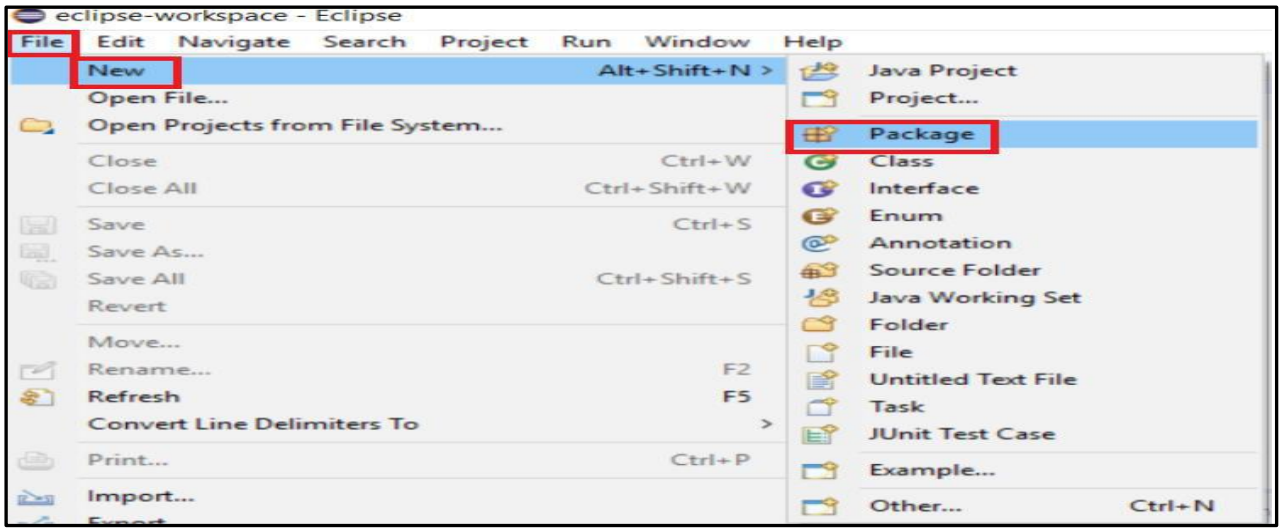
Java Package in Eclipse:

How to create Java Project in Eclipse?

Packages in Java are a group of classes, interfaces and sub-package., which helps in controlling access. To create new Java package in Eclipse is quite easy. You can also create a package like the below,

Step1: Launch Eclipse and go to the File Menu, then

File > New > Package



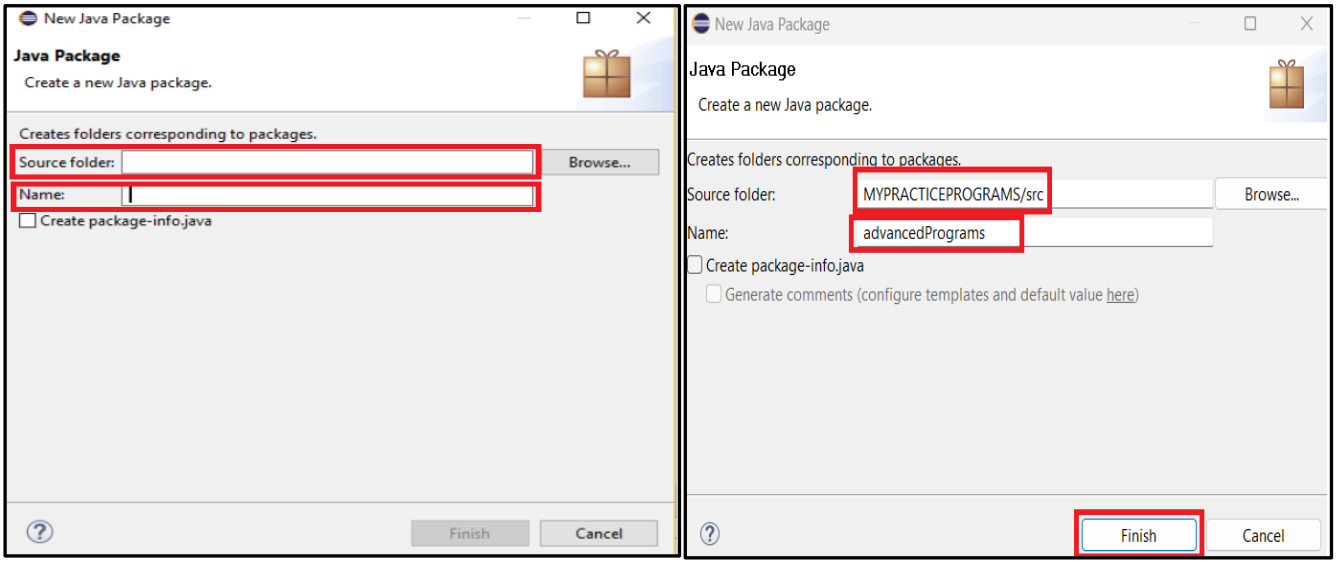
Step2: Now, a dialog box opens up, where you need to fill the source folder of package and the package name.

Here are some of the options provided by the New Java Package wizard in Eclipse,

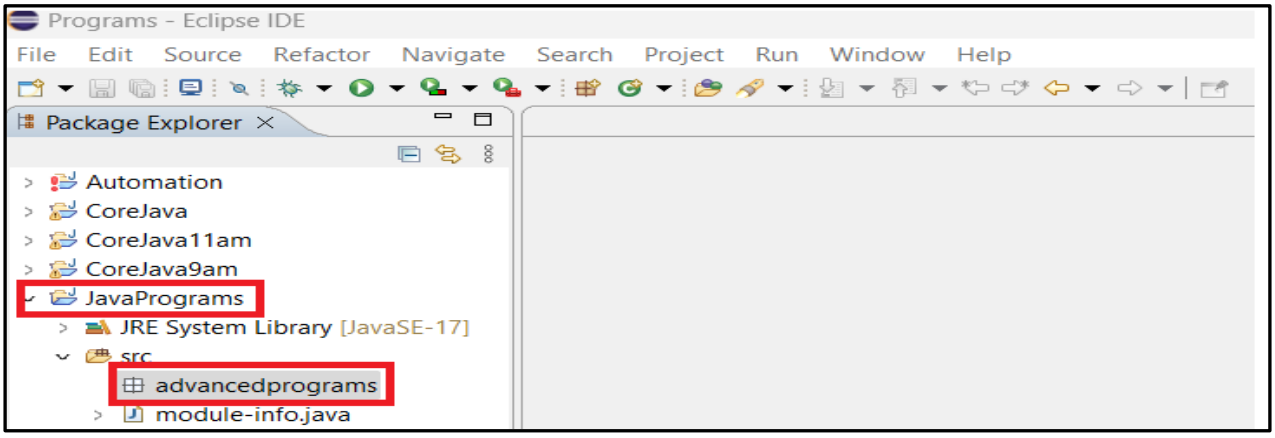
Source Folder: Select where the new package gets saved. The default visible is the source folder, selected when the wizard started.

Name: Here comes the name of the new package you want to add.

Create package-info.java: A separate file package-info.java is to be created or not. It has the package level documentation and annotations.



Step3: After clicking Finish above, the package gets created successfully as shown below.



Java Class in Eclipse:

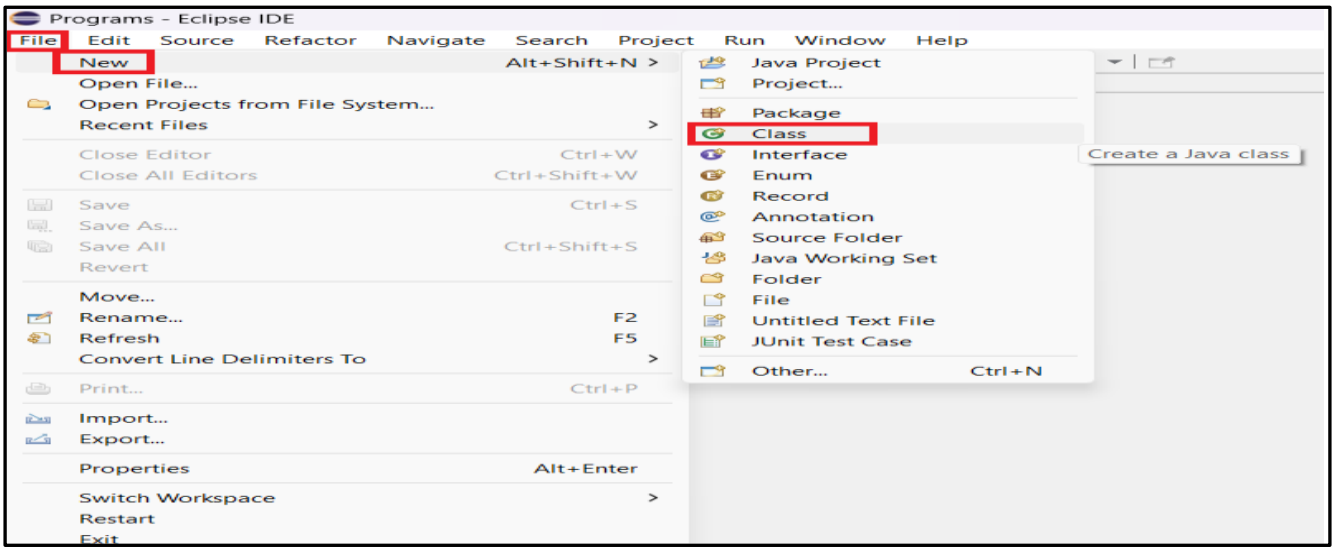
How to create Java Class in Eclipse?

A class in Java is a blueprint to held data in the form of variables and member functions. For creating a new Java class, you need to first create a Java project.

Step1: Launch Eclipse and go to the File Menu, then

File > New > Class

After clicking class above, a new dialog box opens up. It is the wizard to create new Java class in Eclipse.



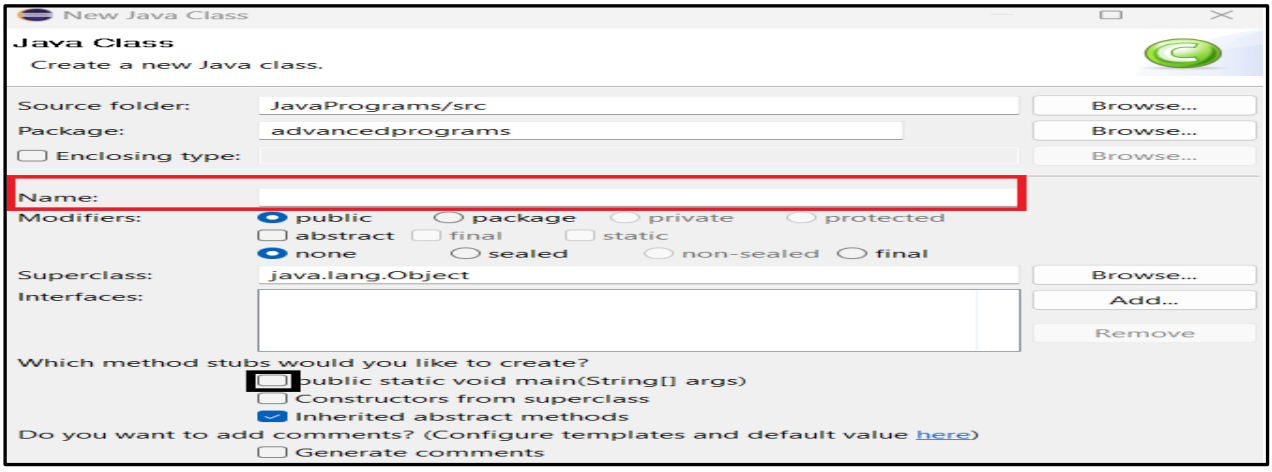
Step2: Here are the options provided by the new Java class wizard,

Source Folder: Enter source folder for the new Java class or click Browse to select from a dialog box.

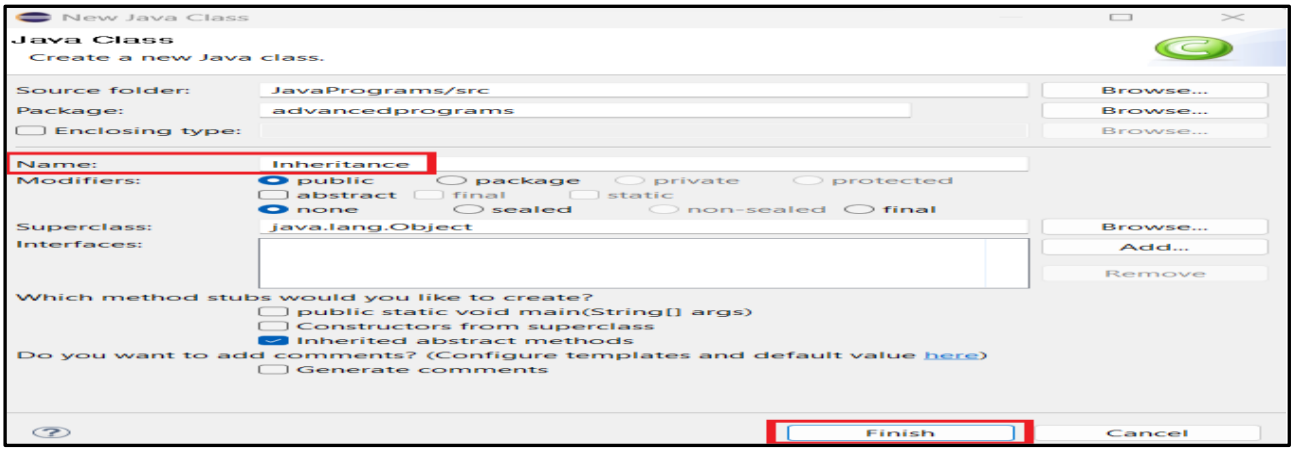
Package: Mention the package to contain the new class. Add the package name or select one by clicking Browser.

Enclosing type: Choose a type to enclose the new class. Select this or the above Package option.

Name:



✓ Here, types the name of the Java class.



Modifiers: Access modifiers such as public, private, protected, abstract, etc. Here, select one of more access modifiers for the new Java class you're adding.

Superclass: Add a superclass for this class.

Interfaces: Add interfaces by clicking Add. These interfaces are the one, which the new class implements.

Which method stubs would you like to create?

Select the method stubs to create in this new class: public static void main (String [] args): A main method stub to the new class.

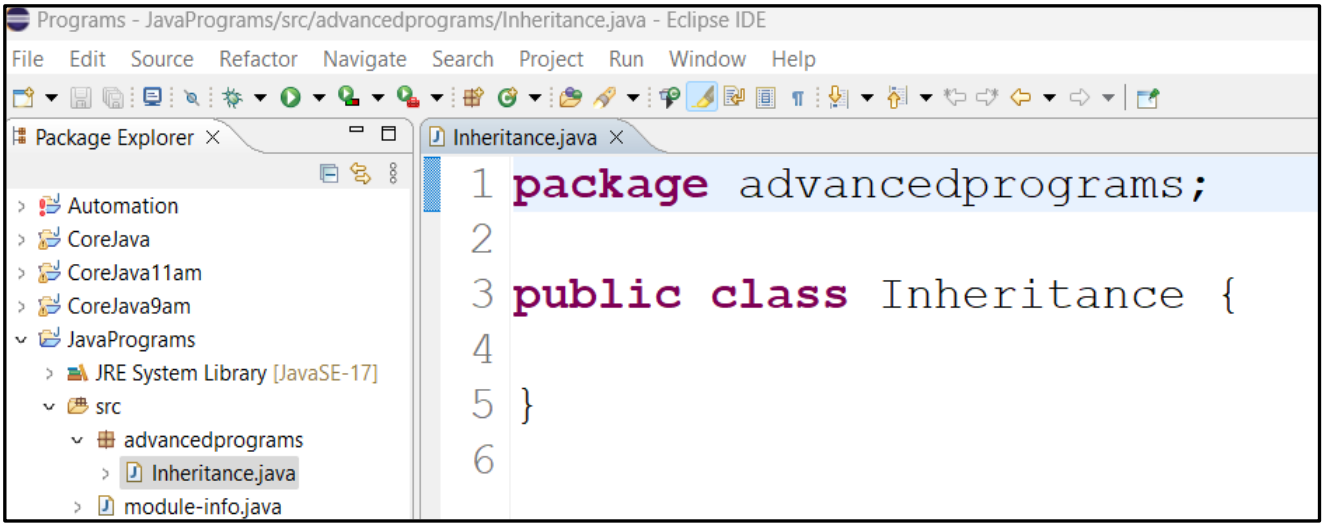
Constructors from superclass: On selecting this, it copies the constructors from the new class's superclass. Adds these stubs to the new class.

Inherited abstract methods: It adds to the new class stubs of any abstract methods from superclass.

Do you want to add comments? The wizard adds comments to the new class, if this option is selected.

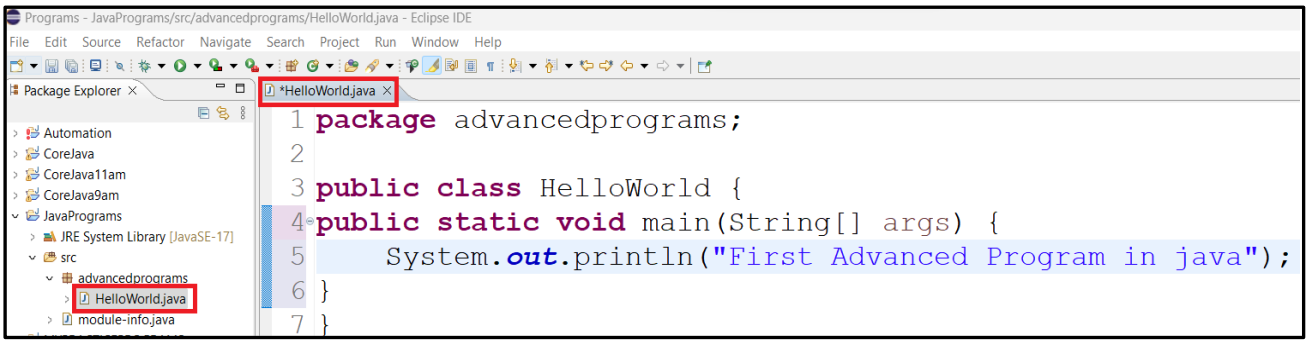
Step3: Click the Finish button.

✓ The newly created class should appear in the Package Explorer view and a java editor instance that allows you to modify the new class.

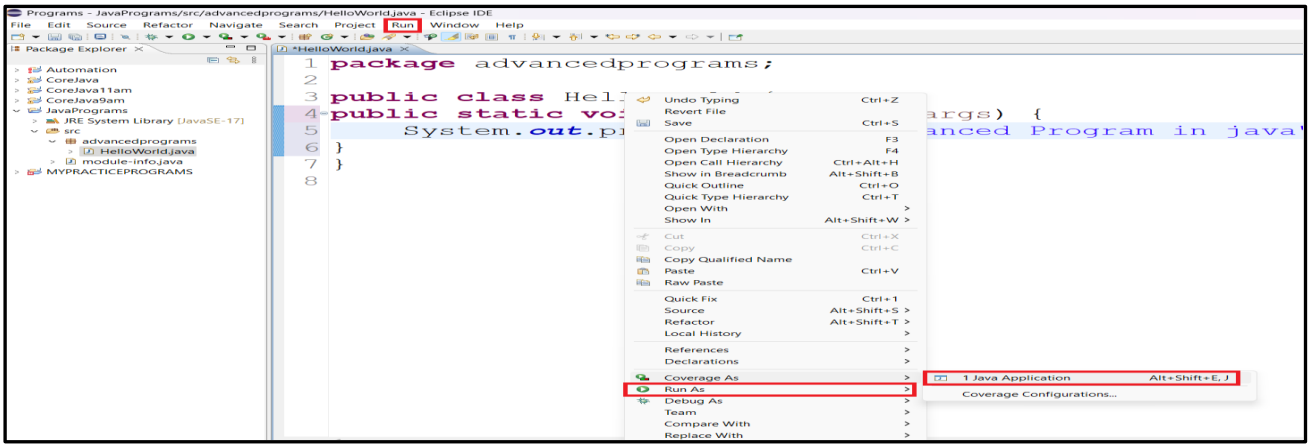


Create First Java Program in Eclipse for Java Programming:

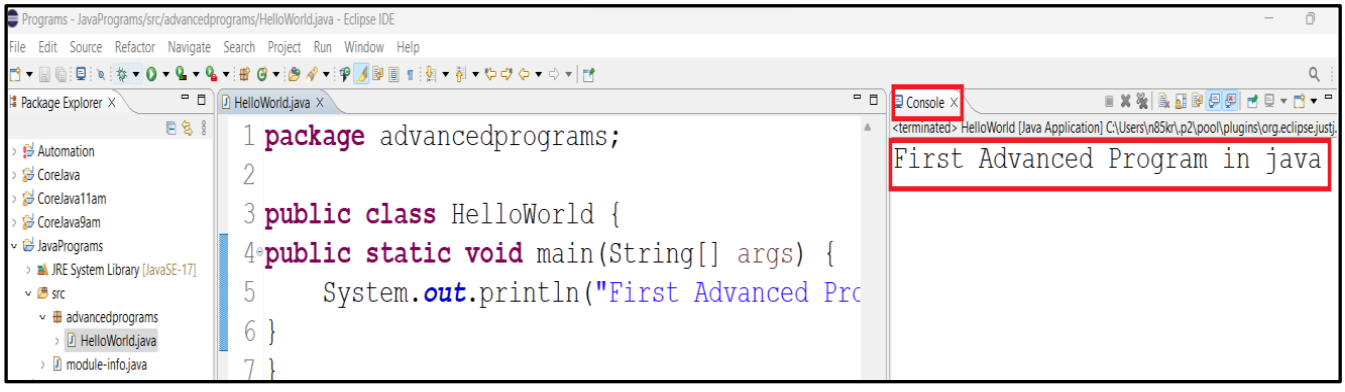
This is how our first program looks now…



✓ Right-click on 'HelloWorld.java' and select from context menu Run As > Java Application.



✓ After running the program, you will get the following output:



Scanner Class:

If we initialize the variables directly in program means setting input values is known as hard code. Hardcode means we cannot alter (change) input values without modifying the program.

How to give different input every time without modifying program?

Java is Providing predefined class called Scanner, With the help of Scanner class method we can enter input values while running the program. It means no need initialize the value in the program (no need to do hard code).

About Scanner class:

Class name: Scanner

Purpose: Read or scan input data, through keyboard we can enter the input data

Type of class: Pre defined (in built)

Methods:

Nature of methods: **non static**

int **nextInt()** ,float **nextFloat()** ,double **nextDouble()**, boolean **nextBoolean()**

To scan **String data we have two methods

next(), **nextLine()**

*** To scan **character** i.e char type of data we do not have any method inside Scanner class,

then how to scan char type of data?

Here to scan char type of data we need to take help from String class. Inside String class, we have a Predefined method.

Method: charAt(index)

charAt() method will return only particular position (index) value.

Note: Memory allocation will be at Run time if we create object then only, we can access the methods.

Method	Description
nextByte()	Accepts a byte
nextShort()	Accepts a short
nextInt()	Accepts an int
nextLong()	Accepts a long
next()	Accepts a single word
nextLine()	Accept a line of String
nextBoolean()	Accepts a boolean
nextFloat()	Accepts a float
nextDouble()	Accepts a double

Steps:

- Create Object
- Import package in which Scanner class is define
- Call methods using object name

Create Object:

Syntax:

classname objectname = new classname(ifparameters);

Scanner scan=new Scanner (System.in);

Here **Scanner** is a class name.

scan is user defined object name.

new is a keyword to allocate memory.

System is a predefined class.

in is input stream, it is a static function.

Input data is supplied using keyboard

** *Always when we use any pre-defined class then we have to import package.

import java.util.Scanner;

Here **import** is a keyword

java.util is package name in which Scanner class is define

Call method by using object name

scan. nextInt(), scan.nextFloat(), scan.nextBoolean, scan.next(), scan.nextInt()

****

Java Tokens

In Java, the program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation. These statements and expressions are made up of tokens. In other words, we can say that the expression and statement is a set of tokens. The tokens are the small building blocks of a Java program that are meaningful to the Java compiler. Further, these two components contain variables, constants, and operators.

Tokens are the various elements in the java program that are identified by Java compiler. A token is the smallest individual element (unit) in a program that is meaningful to the compiler. In simple words, a java program is a group of tokens, comments, and white spaces. A statement consists of variables, constants, operators, keywords, comments, identifiers, etc. When the program is run, comments are stripped and executable statements are executed. The output is called translation unit. A translation unit is a sequence of tokens: symbols, numbers, and words.

Types of Tokens	
I. Keywords **II. Identifiers** **III. Constants/Literals** **IV. Variables** **V. Comments** **VI. Separators** **VII. Operators**	

I. **Java Keywords:**

Java has a set of keywords, also known as reserved words, which cannot be used as variables, methods, classes, or any other identifiers. Keywords are predefined words in Java that have a special meaning to the compiler and act as instructions in a program. Since keywords are reserved by the Java language, they cannot be used as variable names, object names, or class names. If we attempt to use a keyword as an identifier, the compiler will generate a compile-time error.

Example: String this = "Hello World"; // Compile-time error
System.out.println(this);

Explanation: Here, this is a Java keyword, so it cannot be used as a variable name. Hence, the compiler produces an error.

Note: Java has 50 keywords. true, false, and null are literals, and String is a class; therefore, they are not keywords. const and goto are reserved but unused keywords.

There are **50 keywords** in java					
byte	short	int	long	float	double
char	boolean	void	if	else	switch
case	default	for	while	do	break
continue	return	class	interface	enum	extends
implements	new	this	super	instanceof	public
private	protected	abstract	static	final	synchronized
transient	volatile	native	strictfp	try	catch
finally	throw	throws	package	import	const
goto	assert				

II. **Java Identifiers:**

In programming languages, identifiers are used for identification purposes. Identifiers in Java are names that identify elements such as classes, variables, and methods in a program. In other words, an identifier is one that is for naming variables, user-defined methods, classes, objects, parameters, labels, packages, and interfaces in a program. Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume). In Java, an identifier is a sequence of letters, digits, and underscore characters.

For example, Student, main, length, breadth, num1, num2, number, area, and so on are names of things that appear in the program.

In Java programming terminology, such names are called identifiers. They are case sensitive, which means that area, Area, and AREA are not the same. All are different identifiers.

Rules for naming Identifiers in Java:

1. The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), '$'(dollar sign) and '_' (underscore).
2. It must start with a letter, an underscore (_), or a dollar sign ($).
3. For example "Student@" is not a valid java identifier as it contain '@' special character.
4. Identifiers should not start with digits([0-9]). For example "123student" is a not a valid java identifier.
5. Reserved Words can't be used as an identifier. For example "int while = 20;" is an invalid statement as while is a reserved word.
6. It cannot be true, false, or null.
7. There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.

Valid Identifiers	Invalid Identifiers
StudentName	Student Name // contains a space
STUDENTNAME	
Stuname	stu+name // plus sign is not an alphanumeric character
_studentname	
$studentname	student-2 // hyphen is not an alphanumeric character
stu_group_name	stu_group &_marks // ampersand is not an alphanumeric character
stu123	123student // Begins with a digit

Difference Between Identifiers and Variables:

Identifiers	Variables
An identifier is a name given to a variable, function, array, class or structure.	A variable is a name given to a memory location that can hold a value. A variable is a container (storage area) to hold data.
An identifier is used to give a unique name to an entity.	A variable is a unique name to identify a memory location.
All identifiers are not variable.	All variables' names are identifier.
Used to uniquely identify classes, methods, variables, etc.	Hold data values during the execution of a program.
No specific declaration keywords. Can include class names, method names, variable names, etc.	Declared using specific keywords such as int, double, String, etc.
className, methodName, variableName.	int x, double pi, String name.
Identifiers are names given to classes, methods, variables, packages, etc.	Variables are storage locations identified by a name in which you can store data.

III. **Constants [Literals] in Java:**

Constant' word in the English language basically refers to 'a situation that does not change '. In mathematics, there is a concept called a constant, which is a fixed value that cannot be changed. One example of a constant is pi, because it always has the same value, which is 3.14159. It is one of the fundamental concepts of programming. A constant is a data type that substitutes a literal. Constants are used when a specific, unchanging value is used various times during the program. A value which is fixed and does not change during the execution of a program is called constants in java. In other words, Java constants are fixed (known as immutable) data values that cannot be changed.

Constants in Java are used when a 'static 'value or a permanent value for a variable has to be implemented. Java doesn't directly support constants. To make any variable a constant, we must use 'static' and 'final' modifiers.

How To Declare a Constant in Java:

1. To turn an ordinary variable into a constant, you have to use the keyword "final."
2. As a rule, we write constants in capital letters to differentiate them from ordinary variables.
3. If you try to change the constant in the program, javac (the Java Compiler) sends an error message. This happens because you can only assign a value to a constant once.

Syntax	Example
final datatype identifier _name = constant;	final double pi=3.142;
Types of Constants	
1. Integer Constants	2. Real or Floating-Point Constants
3. Character Constants	4. String Constants

1. **Integer Constants:** An integer constant is a sequence of digits without a decimal point. For example, 10 and -200 are integer constants. There are three types of integer constants. They are as follows:

Decimal Integer	Octal Integer	Hexadecimal Integer
Decimal integer is a sequence of digits from 0 to 9. These constants are either positive or negative. Plus, sign is optional.	An octal integer constant is a sequence of any combination of digits from 0 to 7. Octal integer always starts with 0.	Hexadecimal integer constants consist of digits 0 to 9 and alphabets "a" to "f" or "A" to "F". These constants are

For example, 24, -55, 0, and +3425 are valid examples of decimal integer constants in decimal notation. Non-digit characters, embedded spaces, and commas are not allowed between digits. For example, 12 3450, 20.00, $1200 are illegal numbers.	The digits 8 and 9 are not valid in octal notation. The examples of valid integer constants in octal notation are 024, 0, 0578, and 0123.	preceded by 0x. Letter "A" to "F" represents the numbers 10 to 15. The valid examples of hexadecimal integer constants are 0x23, 0x5B, 0x9F, 0x, etc. "x" can also be either small or capital. We rarely use octal and hexadecimal integer numbers system in programming.

2. Real (Floating-Point) Constants: Real constants consist of a sequence of digits with fractional parts or decimal points. These constants are also called floating-point constants. The valid examples of real constants are 2.3, 0.0034, -0.75, 56.7, etc.

These numbers are shown in the decimal point notation. It is also possible that the number may not have digits before decimal point or digits after the decimal point. For example, 205., .45, -.30, etc. A real constants number can also be expressed in exponential or scientific notation. For example, the value 235.569 can also be written as 2.35569e2 in exponential notation. Here, e2 means multiply by 10^2.

3. Single Character Constants: A single character constant (Simply character constant) is a single character enclosed within a pair of single quote. The example of single-character constants are as follows: '5' 'x' ';' ' ' etc. The last constant is blank space. The character constant '5' is not equivalent to the number 5.

4. String Constants: A string constant is a sequence of characters within a pair of double-quotes. The characters can be alphabets, special characters, digits, and blank spaces. The valid examples of string constants are given below:

"Hello Java" "1924" "?…!" "2+7" "X" etc.

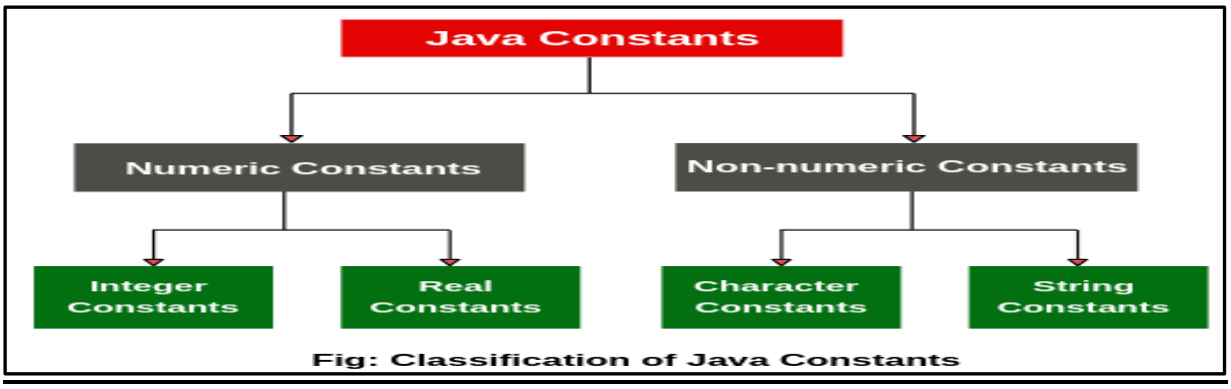


Fig: Classification of Java Constants

Literals (Constants) in Java:

A constant value which can be assigned to the variables is called Literal.

Example: int var = 20;

Here int is the data type, var is a variable name and 20 is a literal or constant value.

Types of Literals in Java:

1. Integral Literals: For the integral data types (byte, short, int, long) the following are various ways to specify Literal value.		
Decimal Literals: The allowed digits are 0 to 9. Example: int x=10;	**Octal Literals:** The allowed digits are 0 to 7. Literal value should be prefixed with 0 (zero). Example: int x= 010;	**Hexadecimal Literals:** The allowed digits are 0 to 9, a to f or A to F. For the extra digits we can use both uppercase and lower case. This is one of few places where the Java language is not case sensitive. Literal value should be prefixed with 0x or 0X

		Example: int x= 0x10; Or int x=0X10;

2. Floating Point Literals:

All floating-point literals are by default double type and hence we cannot assign directly to float type. But we can specify explicitly floating-point literal is of float type by suffixing with f or F. We can specify floating point literal explicitly as double type by suffixing with d or D.

Example:

float f= 123.456; // Compile time exception. Possible Loss of Precision.
float f= 123.456f;
double d= 123.456;
double d= 123.456d;

3. Boolean Literals: The only possible values for the boolean data types are true and false.

Which of the following are valid boolean declarations?

boolean b= 0; // Compile time exception. Incompatible types
boolean b= True; // // Compile time exception. Cannot find symbol.
boolean b= "true"; // Compile time exception. Incompatible types
boolean b= true; // valid
boolean b= false; // valid

4. Char Literals: A char literal can be represented as single character within single quotes.

Ex: char c= 'a';
char c= a; // invalid
char c= 'ab'; // invalid

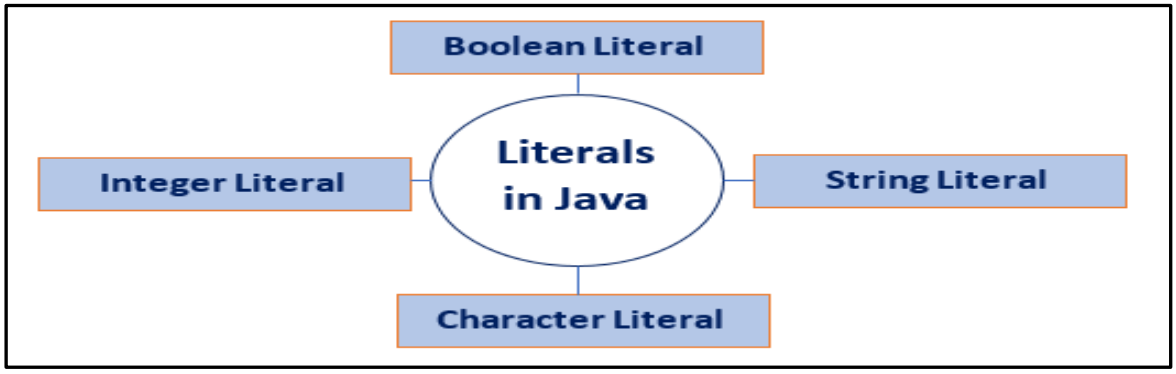
A char literal can be represented as integral literal which represents unicode of that character.

We can specify integral literal either in decimal form or octal form or hexadecimal form but allowed range 0 to 65535.

Ex: char c= 97;
System.out.println(c); // o/p = a
char c= 65535;
char c= 65536;// Compile time exception.
char c= 0xface;
char c= 0642;

5. String Literals: Any sequence of characters within " " (double quotes) is called string literal.

Ex: String name= "Core Java";



Difference between Constants and Literals:

Constants	Literals
Represent fixed values that cannot be changed Actual values used in the program.	Actual values used in the program.

Typically declared using the final keyword.	Directly used in the code without any declaration.
Ex: final int MAX_VALUE = 100;	int number = 42; or "Hello, World!".
Used to define values that should not change.	Used to represent specific values in code.
Variable whose value cannot change once assigned.	Fixed value placed directly in the code.

IV. Variables:

Variable in Java is the basic unit of storage. It acts as a container and is used to hold data values. The values held by the variable can be changed during the execution of the program. Every variable is assigned a data type. Variable, in simpler terms, is a name given to a memory location. Variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variables are containers for storing data values. A variable is a memory location name where information is stored. Information is data it can be Alphabets, numbers, special characters.

A variable is declared by specifying the following **parameters**:

Datatype: Type of data stored in the variable

Variable Name: A unique name of the variable

Value: The initial value stored in the variable

Ex: int age = 50;

float weight = 50.60f;

In the above example, **int** is the data type, **age** is the name given to the variable, and **50** is the value. Similarly, **float** is the data type, **weight** is the variable name and **50.60** is the value.

What is Declaration, Initialization?

Declaration tells about

a. What is the location name (i.e variable name)
b. What type of data it can store

Initialization defines What information we stored in location.

Syntax:

datatype variablename=value;

datatype variablename ---> declaration

variablename=value---> initialization

Variables in Java can be of Different Types:

int: Used to store integer values such as **567**.

float: Stores floating-point numbers such as **29.99f**.

char: Stores single characters, such as '**s**', '**R**'.

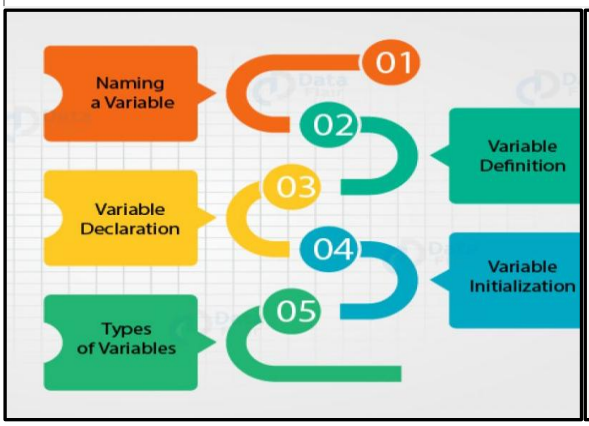
boolean: Stores values that pertain to two states "**True** or **False**"

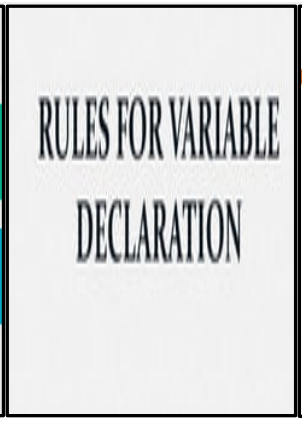
String: Used to store textual matter such as "**Welcome**".

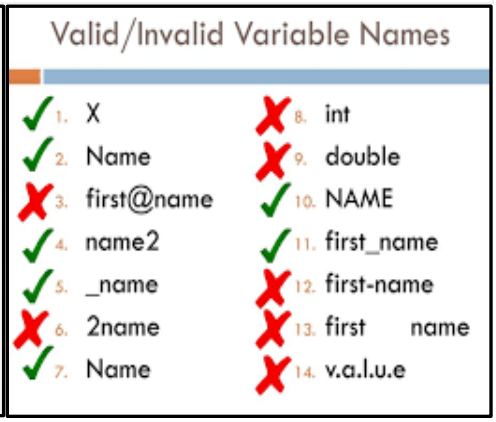
Rules for declaring variable: Variable name is user defined but follows below rules.

Rules	Example
Variable name must start with alphabet	Marks [uppercase: acceptable but shows warning] marks [lowercase: valid]
Do not start with numbers or special characters	9marks=100 --> invalid *marks=100--> invalid marks9=100--> valid
Do not contain any spaces	total marks=420; ---> invalid totalMarks=420; valid

	total_marks=420; valid
Do not use keyword as variable name	char int='A'; invalid [int is a keyword] char ch='A'; valid.







Syntax: For creating variable
datatype variable name=value;
datatype variablename ---> declaration
variablename=value---> initialization

Example1	**Example2**	**Example3**
Create a variable called name of type String and assign it the value "Hyderabad": String city = " Hyderabad "; System.out.println(city);	Create a variable called data of type int and assign it the value 15: int data = 15; System.out.println(data);	How to declare variables of other types: float percentage = 85.99f; char grade = 'D'; boolean result = true;
Note: You can also declare a variable without assigning the value, and assign the value later. int data; data = 15; **Note:** That if you assign a new value to an existing variable, it will overwrite the previous value: Change the value of data from 15 to 20: int data = 15; data = 20; // data is now 20 System.out.println(data);		

Scope of Variable:
Each variable is defined and can be used within its scope and determines that wherein the program this variable is available to use. The scope means the lifetime of that variable. It means the variable can only be accessed or visible within its scope. The scope of variables can be defined with their declaration, and variables are declared mainly in two ways:

1. **Global Variable:**
   - Global variables are those variables which are declared outside of all the functions or block and can be accessed globally in a program.
   - It can be accessed by any function present in the program.
   - Once we declare a global variable, its value can be varied as used with different functions.
   - The lifetime of the global variable exists till the program executes. These variables are stored in fixed memory locations given by the compiler and do not automatically clean up.
   - Global variables are mostly used in programming and useful for cases where all the functions need to access the same data.
2. **Local Variable:**

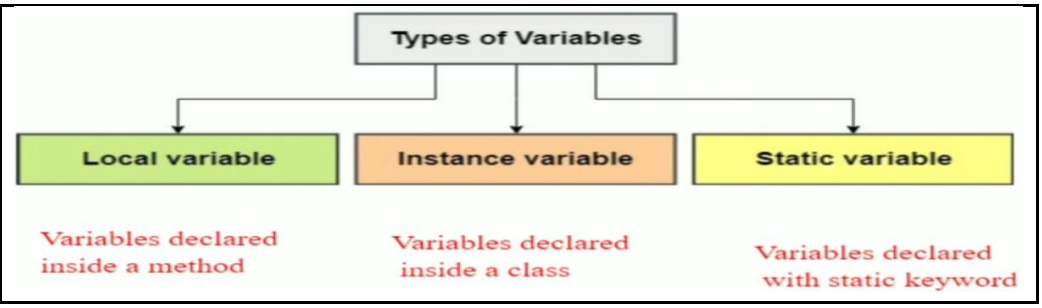
- Variables that are declared within or inside a function block are known as Local variables.
- These variables can only be accessed within the function in which they are declared.
- The lifetime of the local variable is within its function only, which means the variable exists till the function executes. Once function execution is completed, local variables are destroyed and no longer exist outside the function.
- The reason for the limited scope of local variables is that local variables are stored in the stack, which is dynamic in nature and automatically cleans up the data stored within it.
- But by making the variable static with "static" keyword, we can retain the value of local variable.

Difference Between Global Variable and Local Variable:

Global Variable	Local Variable
Global variables are declared outside all the function blocks.	Local Variables are declared within a function block.
The scope remains throughout the program.	The scope is limited and remains within the function only in which they are declared.
Any change in global variable affects the whole program, wherever it is being used.	Any change in the local variable does not affect other functions of the program.
A global variable exists in the program for the entire time the program is executed.	A local variable is created when the function is executed, and once the execution is finished, the variable is destroyed.
If the global variable is not initialized, it takes zero by default.	If the local variable is not initialized, it takes the garbage value by default.
We cannot declare many variables with the same name.	We can declare various variables with the same name but in other functions.

Types of Variables: There are 4 types of variables in Java:

1. Local variable
2. Instance variable
3. Static variable
4. Final Variables



1. Local Variable:

A variable declared inside the class and inside the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword. Local variable can access directly. Local variables prohibit the use of access modifiers. These variables can be accessed only within the particular block.

Example: int id = 0; // local variable id

2. Instance Variable:

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static. It is called an instance variable because its value is instance-specific and is not shared among instances. Instance variable can access by using object only. Access Modifiers can be used for instance variables. When no modifier is specified, the default modifier is used. Instance Variables have default values, **0 for numbers**, **false for Boolean**, and **null** for **String** object references. Instance variables that are declared in a class and not inside any function or main method.

Example: int price;

Note: Local variable and Instance variable also called as non-static variables.

3. Static variable:

A variable that is declared as static is called a static variable. They are similar in nature to Instance Variables. The major difference is that they are declared using the static keyword and only a single copy of a static variable per class is allowed. Static variable declared using static keyword. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory. Static variable can access directly. To access static variables, creating an object of that class is not necessary.

Example: public static double salary; // static variable salary

Note: Instance variable and Static variable also called as Global variables.

4. Final Variable:

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example: final int data = 15;

data = 20; // will generate an error: cannot assign a value to a final variable.

****

Data Types in Java

In Java, data types are used to specify the type of data that a variable can hold. In Java, data types play a crucial role in defining the kind of information that can be stored and manipulated. Data types refer to the different sizes and values that can be stored in the variable. A data type is a classification of data. It tells the compiler or interpreter how the programmer aims to use the variables. Data types in Java specify how memory stores the values of the variable. Each variable has a data type that decides the value the variable will hold. In Java, data types are used to define the type of data that a variable can hold. Java is a statically-typed language, which means that all variables must be declared with a specific data type before they can be used.

Can we store any type of data in location or variable?

Yes, we can store any type of data means number, alphabets, special characters, decimal numbers. There are no restrictions you can store any type of data in location.

What is the problem if we do not restrict data to store in a location?

Example:

Let's add two numbers

store number1 i.e 10 in location a--> a=10

store number2 i.e 20 in location b--> b=20

store result after calculating in location c--> c=a+b

No error found because we can add two numbers, but if we enter input as other than number , example it can be alphabet or special character, therefore we cannot add number with alphabets or special character , then we have to face unexpected results or errors.

To overcome this, we need someone who cross verify data before storing into location.

** To cross verify the data we have to use Datatypes.

Data Types in Java:

Data types specify the different sizes and values that can be stored in the variable. what type of values that can be stored into memory location. All datatypes are **Keywords**, they are in small letters.

Data types in Java are divided into **TWO** categories:

1.Primitive data types **2. Non-Primitive data types**

1. **Primitive Data Types:**

Primitive data types are the fundamental building blocks of data manipulation. These are the most basic data types available in Java language. Primitive type of data it is also known as built-in data types. Primitive

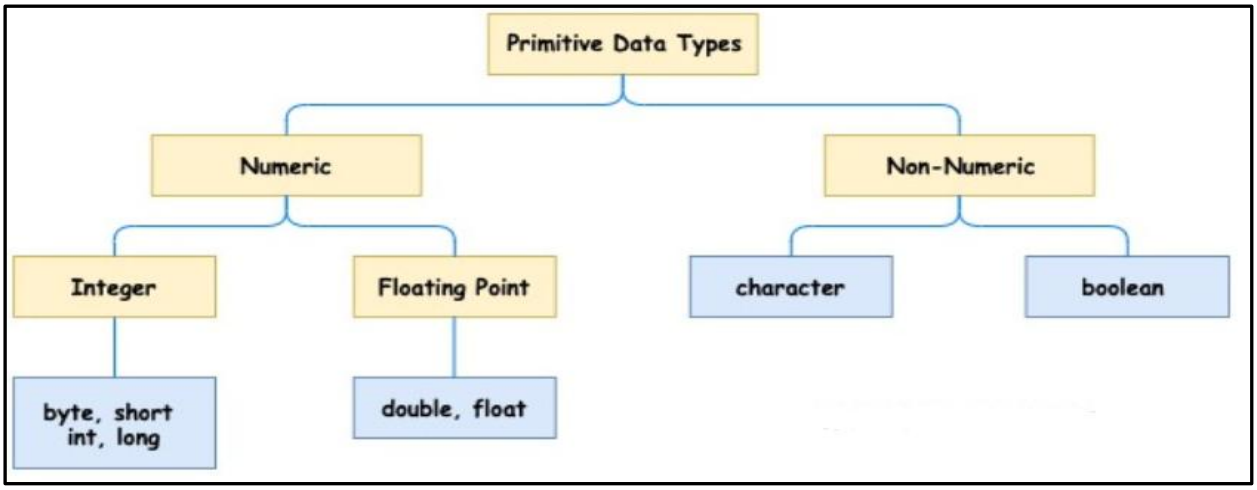
datatype is not user define i.e. Programmers cannot develop. In Primitive type of data, we can store only one value at a time, it known as general or fundamental types of data. Primitive data types serve only one purpose containing pure, simple values of a certain kind. They are reserved keywords in Java. Because they are keywords, they cannot be used as variable names.

- ✓ Primitive data types store the actual value directly in memory.
- ✓ Primitive data types have default values if not explicitly initialized.
- ✓ Primitive data types have fixed sizes defined by the language.
- ✓ Operations on primitive data types can be performed directly.
- ✓ When passing a primitive data type as a method argument, a copy of the value is passed.

Primitive data types are divided in **Two** types

Primitive Data Types	I. Numeric	1. Integer	a) byte
			b) short
			c) int
			a) long
		2. Floating point	b) float
			c) double
	II. Non-Numeric	3. Characters	d) char
		4. Boolean	e) boolean

There are **8** Primitive data types in Java –byte, int, short, long, float, double boolean and char.



Numeric Data Types:

1. Integer Data Types:

In Java, the integer data types are used to represent whole numbers without any fractional component. Integer types store whole numbers, positive or negative (such as 123 or -456), without decimals.

There are four types: **byte, short, int and long**.

a. Byte: The byte is the smallest data type among all the integer data types. This can be used instead of int or other integer types to save memory because a byte is **4** times smaller than an integer. It is an **8**-bit signed two's complement integer. The byte data type can store whole numbers. Its minimum value is **-128** and maximum value is **127**. Its default value is **0**. It is the first data type with least memory size allocation which can be used for numbers of small ranges. Keyword is **byte**.

Syntax: byte variable;

Ex: byte a = 10, byte b = -20;

b. Short: The short data type is a **16**-bit signed two's complement integer. A short data type is 2 times smaller than an integer. Its minimum value is **-32,768** and maximum value is **32,767**. Its default

value is **0**. The short data type can also be used to save memory just like byte data type. Keyword is **short**.

Syntax: short variable;

Ex: short s = 10000, short r = -5000;

c. Int: The int data type is a **32**-bit signed two's complement integer. The int data type is generally used as a default data type for integral values unless if there is no problem about memory. Its minimum value is - **2,147,483,648(-2^31)** and maximum value is **2,147,483,647(2^31 -1)**. Its default value is **0**. Use the Integer class to use int data type as an unsigned integer. The int data type is the preferred data type when we create variables with a numeric value. Keyword is **int**.

Syntax: int variable;

Ex: int a = 100000, int b = -200000;

d. Long: The long data type is a **64**-bit two's complement integer. Its minimum value is- **9,223,372,036,854,775,808(-2^63)** and maximum value is **9,223,372,036,854,775,807(2^63 -1)**. Its default value is **0L**. This data type ends with '**L**' or '**l**'. The long data type is used when you need a range of values more than those provided by int. Keyword is **long**.

Syntax: long variable;

Ex: long a = 100000L, long b = -200000L;

2. Floating Point Data Types:

Floating point data types are a fundamental part of programming languages, used to represent decimal numbers with fractional parts. They are essential for handling real numbers in computations where precision matters. Floating point numbers are also called floating point literals in Java. For example, floating point or decimal numbers are used when expressing currency (dollars and cents), measurements (feet and inches), and time (hours, minutes, and seconds). For calculations using decimal values, Java supports two basic data types: **float** and **double**. Both float and double data types can store positive and negative values.

a. Float: The float data type can store fractional numbers. You should use a floating-point type whenever you need a number with a decimal, such as 9.99 or 3.14515. A float data type in Java stores a decimal value with 6-7 total digits of precision. So, for example, 12.12345 can be saved as a float, but 12.123456789 can't be saved as a float. The float data type is a single-precision 32-bit IEEE 754 floating point. Can have a 7-digit decimal precision. Its default value is 0.0F. Note that you should end the value with an "f" or 'F' for float. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating-point numbers. It is not used for precise data such as currency or research data. Float data type is used when you want to save memory and when calculations don't require more than 6 or 7 digits of precision.

Syntax: float variable;

Ex: float f1 = 234.5f;

b. Double: The double data types can store fractional numbers. The double data type is similar to float. The difference between the two is that is double twice the float in the case of decimal precision. It means that it gives 15-16 decimal digits precision. It consumes more memory in comparison to the float data type. The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d. For decimal values, this data type is generally the default choice. It is optional to add suffix d or D.

Syntax: double variable;

Ex: double d1 = 12.3;

Note: Even though there are many numeric types in Java, the most used for numbers are **int** (for whole numbers) and **double** (for floating point numbers).

Non-Numeric Data Types:

a. Characters (char):

The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c'. To declare and use a char in Java, you use the char keyword before a variable name. It's used to store any character from the Unicode character set, which includes letters, digits, symbols. However, chars in Java can represent a single character only. If you need to work with multiple characters, you'll need to use a string or an array of chars. The CHAR data type stores character data in a fixed-length field. Also, the standard ASCII characters range from 0 to 127. In Java, char is short for character. It's 16 bits in size - double that of a byte.

Syntax: char variable;

Ex: char letter = 'A'

b. Boolean (boolean):

In Java, the boolean data type represents a simple data type that can have one of two values: true or false. A boolean type is declared with the boolean keyword and can only take the values true or false. It is used to store logical values and is commonly used in conditional statements, such as if-else statements and loops, as well as in boolean expressions. Values of type boolean are not converted implicitly or explicitly (with casts) to any other type. A Boolean expression returns a boolean value: true or false. This data type is used for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.

Syntax: boolean variable;

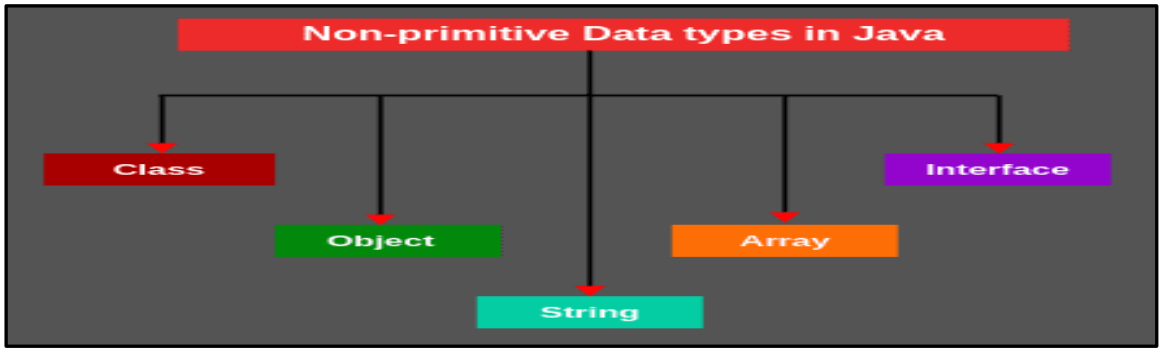
Ex: boolean result = false;

Primitive Type Keyword

Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined*	true or false	false

2. Non-Primitive Data Types:

Non-primitive data types in Java are not predefined. Non-primitive types are created by the programmer and is not defined by Java (except for String). Non-Primitive Data Type also known as Reference Data Types or derived data types or advanced datatypes. The Reference Data Types will contain a memory address of variable values because the reference types won't store the variable value directly in memory. They are **strings**, **arrays**, **classes** and **objects** etc…



a. Class: Class is a template or blue print. Class is logical existence. Class is a collection of objects. class is a collection of variables and methods. A document in which features and functions are defined i.e. known as Class. Class is a keyword always write in lowercase only. class follows with Classname and no semicolon and follows with body i.e. open brace and close brace ({}) inside that will write code.

Syntax: class ClassName

Ex: class Student_Details

b. Object: A thing which is having identity, state, behaviour known as Object. Object is an instance of class. Object is physical existence. Object inherits features of a class. Classes and objects are the two main aspects of object-oriented programming. So, a class is a template for objects, and an object is an instance of a class. When the individual objects are created, they inherit all the variables and methods from the class.

Syntax: ClassName object = new ClassName();

Ex: Student s1=new Student();

c. Strings: Strings are used for storing text. A String variable contains a collection of characters surrounded by double quotes.

Syntax: <String_Type> <string_variable> = "<sequence_of_string>";

Ex: String greeting = "Hello";

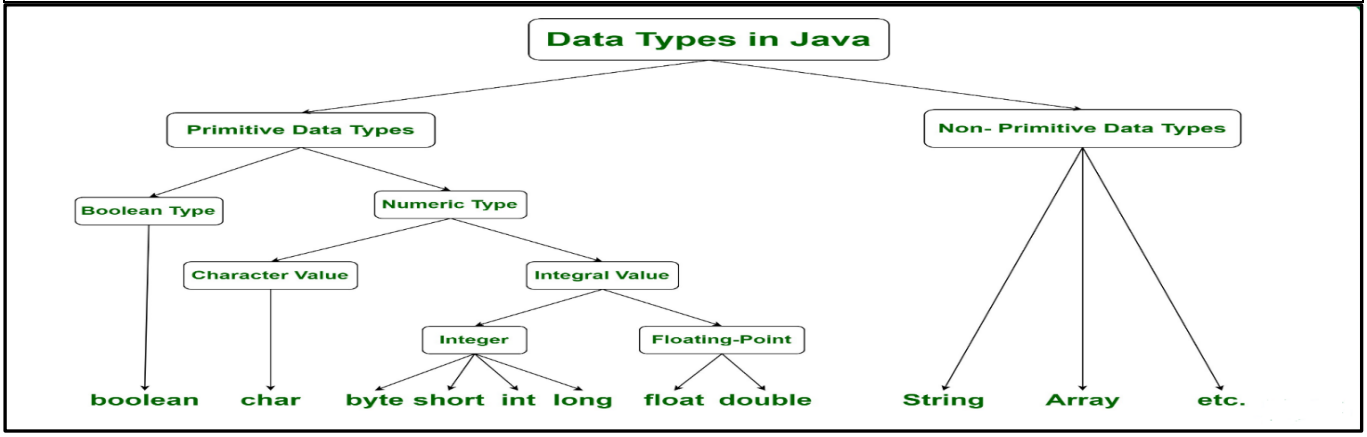
Here String is data type, greeting is a variable and Hello is text.

The difference between a char and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a char is a collection of separate char type entities.

d. Array: Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with square brackets: int[] marks; We have now declared a variable that holds an array of integer.

Syntax: datatype [] arrayname = {value1, value 2, value 3, value 4, value 5};

Ex: int [] numbers = {1, 2, 3, 4, 5};



Main Difference Between Primitive and Non-Primitive data Types:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types start with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

****

Comments in Java

It's very often necessary to add some comments when you're writing a program. Comments in Java are the statements that are not executed by the compiler and interpreter. It can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for a specific time.

For Example:

- This block code is responsible for…
- This variable is responsible for… etc.

It's a very good practice to add comments. It will always be easy for you to remember what does what in your program. It often happens that a person writes a code and then opens it a month later and can't believe that they were the one who wrote it. But if you leave comments, the shock won't be as great. And anyone who works on your code after you will be grateful. After all, you might not be at your colleague's side when they read your code. If you don't leave comments, they may be confused and unable to understand why you wrote what you wrote. That's why you should add comments – to take care of yourself and future programmers.

In Java there are **Three Types** of comments:

A. Single-Line Comments:

Single-line comments start with two forward slashes (//). Any text between // and the end of the line is ignored by Java (will not be executed). This example uses a single-line comment before a line of code.

Syntax: //Comments here (Text in this line only is considered as comment).

Ex: **// Prints the below text.**

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

```java
public class Demo {
    public static void main(String[] args) {
        // The below print statement prints the text
        System.out.println("QAFox");
        // The below print statement prints the number
        System.out.println(9);
    }
}
```

B. Multi-Line Comments:

Multi-line comments start with /* and ends with */. Any text between /* and */ will be ignored by Java. Single line comments can be tedious to write since we have to give '//' at every line. So, to overcome this multi-line comments can be used.

Syntax:

/*Comment starts
continues
continues
.
.
.
Comment ends*/

Example:

```
class Multilinecomments
{
public static void main(String args[])
{
System.out.println("Multi line comments below");
/*Comment line 1
Comment line 2
Comment line 3*/
}
}
```

```java
public class Demo {
    /* In this Java program, I am demonstrating
       different types of comments in Java
       i.e. Single line and Multi-line comments
     */
    public static void main(String[] args) {
```

C. Documentation Comments:

/** documentation */

This is a documentation comment and in general it's called doc comment. The JDK javadoc tool uses doc comments when preparing automatically generated documentation. This type of comment is used generally when writing code for a project/software package, since it helps to generate a documentation page for reference, which can be used for getting information about methods present, its parameters, etc.

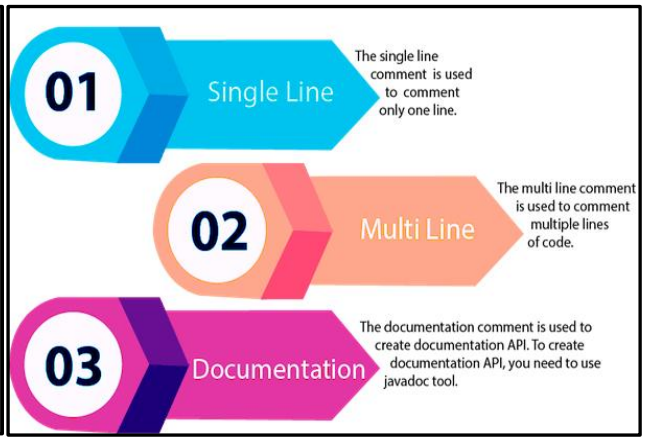
Example:

```
/**
* The HelloWorld program implements an application that
```

* simply displays "Hello World!" to the standard output.
* @author Zara Ali
* @version 1.0
* @since 2014-03-31
*/

```java
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {

   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```



****

Separators in Java

Separators help define the structure of a program. In Java, there are few characters used as separators. Separators are used to separate one programming element from other. Separators in java are nothing but some symbols or characters that are used to structure a java program. The most commonly used separator in java is a semicolon (;).

Let us see a basic program in java:

```java
class javaexample {
    Run | Debug
    public static void main(String[] args) {
        System.out.println(x: "Hello World");
        int age = 20;
        if (age > 30) {
            System.out.println(x: "Im greater than 30");
        } else {
            System.out.println(x: "Im smaller than 30");
        }
    }
}
```

In the above example, we can see some symbols like ; : . } { (). These are nothing but separators in java.

Symbol of Separator	Name of Separator	Use of Separator

{ }	Braces	Used to define a block of code, for classes, methods and local scopes Used to contain the value of automatically initialised array
()	Parenthesis	Parentheses are used to define the parameters of the methods in java. Also used in control statements & type casting. Parentheses are also the most commonly used for containing the expression in the if-else statement.
[]	Brackets	Brackets are used in array declaration in java.
;	Semicolon	A semicolon is used to terminate the statement in Java. Whenever we encounter a semicolon after a statement, it means that the statement has ended and the next statement is a different one.
,	Comma	It is used to select a field from an object. We often declare the variables in a single line. The comma is used here to distinguish these variable names from each other.
.	Period	It is used to separate the package name from sub-package name & class name. It is also used to separate variable or method from its object or instance.
:	Colon	Colon is used to create a method reference. It is also used for the constructor reference. We often use colons after labels.

Escape Sequence (or) Backslash Character Constants in Java:

A character with a backslash (\) just before it is an escape sequence or escape character. We use escape characters to perform some specific tasks. Each escape character is a valid character literal. It may include letters, numerals, punctuations, etc. Remember that escape characters must be enclosed in quotation marks (""). These are the valid character literals. Java provides some special backslash character constants that are used in output methods. For example, the symbol 'n' which stands for newline character.

Escape Characters	Description
\t	It is used to insert a tab in the text at this point.
\'	It is used to insert a single quote character in the text at this point.
\"	It is used to insert a double quote character in the text at this point.
\r	It is used to insert a carriage return in the text at this point.
\\	It is used to insert a backslash character in the text at this point.
\n	It is used to insert a new line in the text at this point.
\f	It is used to insert a form feed in the text at this point.
\b	It is used to insert a backspace in the text at this point.

****

Errors in Java

Rarely does a program run successfully at its very first attempt. It is very common to make mistakes while developing as well as typing a program. A mistake might lead to an error causing the program to produce unexpected results. Errors can make a program go wrong. An error may terminate the execution of the program or may produce an incorrect output or even may cause the system to crash. It is important to detect and manage properly all the possible error condition in the program so that the program will not terminate/crash during execution.

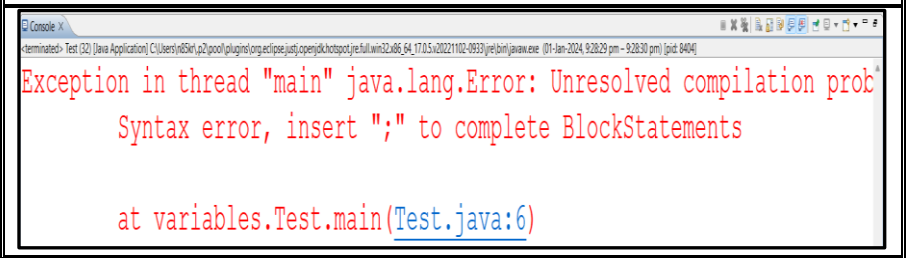
Error: A mistake done by the programmer is known as error (or) A mistake might lead to an error causing the program to produce unexpected results. Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors prevent or prohibit the program from getting compiled or executed. Thus, errors should be removed before compiling and executing. An error may produce an incorrect output or may terminate the execution of the program abruptly (Unexpectedly).

Types of Errors:

1. **Compile Time Errors.** **2. Runtime Errors.**

1. **Compile Time Errors: [**Syntax Errors]

All syntax errors are detected and displayed by the Java compiler and hence these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. Therefore, it is necessary that we fix all the errors before we can successfully compile and run the program. Compile Time Errors are those errors which prevent the code from running because of an incorrect syntax such as a missing semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the java compiler and an error message is displayed on the screen while compiling. Compile Time Errors are sometimes also referred to as Syntax errors. These kinds of errors are easy to spot and rectify because the java compiler finds them for you. The compiler will tell you which piece of code in the program got in trouble and its best guess as to what you did wrong. Usually, the compiler indicates the exact line where the error is, or sometimes the line just before it, however, if the problem is with incorrectly nested braces, the actual error may be at the beginning of the block. In effect, syntax errors represent grammatical errors in the use of the programming language. All syntax errors will be detected and displayed by the java compiler these errors are known as compile time errors.

Example	OutPut
class Sample { public static void main(String args[]) { Ststem.out.println("Java Program") **//Missing semicolon** } }	
The Java compiler does a nice job of telling us where the errors have occurred in the program. For example, if we have missed the semicolon at the end of print statement in Program , the following message will appear on the screen. We can now go to the appropriate line, correct an error and recompile the program.	

Sometimes, a single error may be the source of multiple errors later in the compilation. For example, use of an undeclared variable in several places will cause a series of errors of type “undefined variable”. In such case, we should consider the earliest errors as the major source of problem. Once we fix an error, we should recompile the program and look for other errors. Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find, and we may have to check code word by word.

The most common **Compile Time Errors** are	
• Missing semicolons	• Missing double quotes in strings
• Missing (or mismatch of) brackets in classes and methods	• Using undeclared variables
• Misspelling of keywords and identifiers	• Use of = in place of == operator and so on.
• Missing open and close braces	• Duplicate user defined names

2. **Run Time Errors:**

Sometimes, a program may compile successfully creating .class file but it may not run properly. Such programs may produce incorrect output due to wrong logic or may terminate due to errors such as stack overflow. A program may compile successfully creating the .class file but may not run properly. The errors which occur at run time are known as runtime errors. Run Time errors occur or we can say, are detected during the execution of the program. Sometimes these are discovered when the user enters an invalid data or data which is not relevant. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM (Java Virtual Machine) that detects it while the program is running. To handle the error during the run time we can put our error code inside

the try block and catch the error inside the catch block. Most common runtime errors are dividing an integer by zero, array index out of bounds etc...

Most common Run-Time Errors
▪ Dividing an integer by zero
▪ Accessing an element that is out of the bounds of an array
▪ Trying to store a value into an array of an incompatible class
▪ Passing a parameter that is not in a valid range or value for a method
▪ Trying to illegally change the state of a thread
▪ Attempting to use a negative size for an array
▪ Using a null object reference as a legitimate object reference to access a method or a variable
▪ Converting invalid string to a number

Division by Zero:

Example	OutPut
class Error2 { public static void main(String[] args) { int x = 10; int y = 0; int a = x/y; //Division by zero System.out.println("a=" +a); } }	Exception in thread "main" java.lang.ArithmeticException: / by zero at Test.main(Test.java:10)
Above Program is syntactically correct and therefore does not cause any problem during compilation. However, during execution, it displays the following message and stops without executing remaining statements. When Java run-time tries to execute a division by zero, it generates an error condition, which causes the program to stop after displaying an appropriate message.	

To solve Runtime errors in java we are using **Exception Handling**.

Exception: An exception is a condition caused by a run-time error in the program. When the Java interpreter encounters an error, such as dividing an integer by zero, it creates and throws an exception object (i.e., informs us that an error has occurred). If the exception object is not caught and handled properly, the interpreter will display an error message as shown in the output of Program and will terminate the program. In Java, exceptions are used to handle runtime errors that occur in the middle of the execution of a program. An exception is an object that represents an error or an abnormal condition that has occurred in the program. The try-catch block is used to handle runtime errors in Java. The try block contains the code that may cause an exception, and the catch block contains the code that handles the exception. Runtime errors are difficult to debug, as they may not occur immediately and may cause the entire program to hang. An Exception is a condition that is called by a runtime error in program.

```
public class exceptionExample {
   public static void main(String[] args) {
      try {
         int num1 = 12;
         int num2 = 0;
         int result = num1 / num2; // This line of code will show an ArithmeticException
         System.out.println(result);
      } catch (ArithmeticException e) {
         System.out.println("Error occurred: " + e.getMessage());
      }
   }
}
```

In the above example, we are trying to divide an integer by zero, which is not a valid operation and will result in an 'ArithmeticException'. To handle this exception, we have enclosed the potentially problematic

code in a 'try' block and caught the exception in a 'catch' block. The catch block will execute if an exception occurs and will print an error message along with the exception message.

Examples of Runtime Errors:

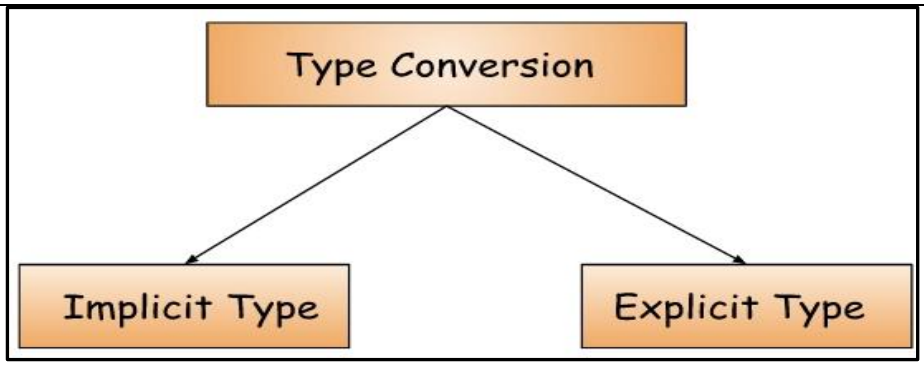
▪ **ArithmeticException**: This error occurs when a program attempts to perform an illegal arithmetic operation, such as dividing by zero. **Ex:** int num = 5 / 0; // This will throw an ArithmeticException because dividing by zero is not allowed.
▪ **ArrayIndexOutOfBoundsException:** This error occurs when a program attempts to access an array element that is outside of the array bounds. **Ex:** int[] arr = new int[3]; int num = arr[3]; // This will throw ArrayIndexOutOfBoundsException.
▪ **NullPointerException:** This error occurs when a program attempts to use a null object reference. **Ex:** String str = null; int length = str.length(); // This will throw a NullPointerException because str is null.
▪ **OutOfMemoryError:** This error occurs when a program runs out of memory.

****

Type Casting / Type Conversion in Java

In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. Type casting is when you assign a value of one primitive data type to another type. A type cast is basically a conversion from one type to another.

In Java, there are two types of type conversions:

1. Implicit Type Conversion (Automatically)	
2. Explicit Type Conversion (Manually)	

1. **Implicit Type Conversion (Widening):**

Implicit Type Conversion Also known as Automatic type conversion. converting a smaller type to a larger type size. Done by the compiler on its own, without any external trigger from the user. All the data types of the variables are upgraded to the data type of the variable with largest data type.

Like byte -> short -> char -> int -> long -> float -> double

Example1: int num1=88; [Output: Before conversion, the value is: 88]

float num2=num1; [Output: After conversion, the value is: 88.0]

Example2: char ch='9'; [Output: Before conversion, the value is: ch = 9]

int num=ch; [Output: After conversion, the value is: num = 57] (ASCII)

2. **Explicit Type Conversion (Narrowing):**

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type. converting a larger type to a smaller size type. This is done by explicitly(manually) defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

Like double -> float -> long -> int -> char -> short -> byte

Syntax: (type) expression (or) variable

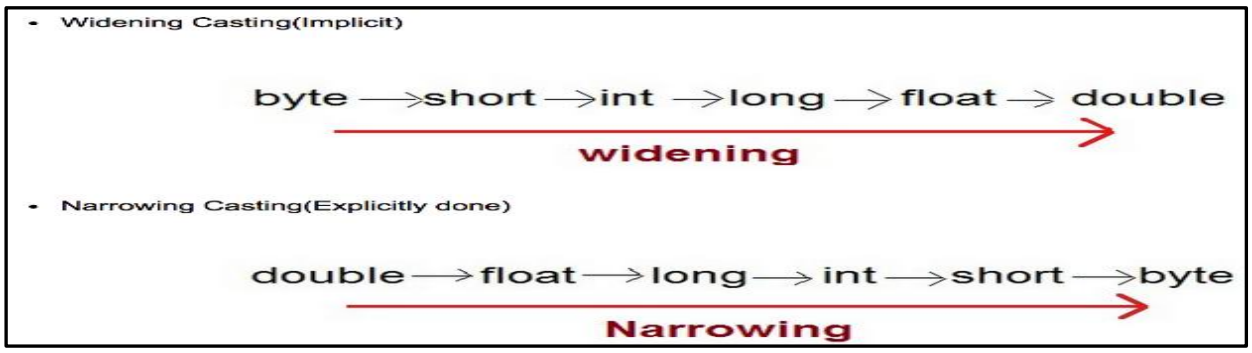
where type indicates the data type to which the final result is converted.

Example1: double myDouble = 9.78d; [Output: Before conversion, the value is: 9.78]

int myInt = (int) myDouble; [Output: After conversion, the value is: 9]

Example2: int j = 75; [Output: Before conversion, the value is: J = 75]

char c= (char)j; [Output: After conversion, the value is: C = K]



Advantages of Type Conversion:

- Type conversion provides flexibility in handling different data types within a program.
- It allows you to perform operations and comparisons between variables of different data types.
- Type conversion can help catch and handle errors related to incompatible data types.
- Java supports automatic type conversion (widening conversion) and explicit type conversion (narrowing conversion), providing developers with a range of options for managing data types efficiently.
- It allows you to validate and convert input data to the expected type, reducing the likelihood of runtime errors.

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Why java uses Unicode System?

Before Unicode, there were many language standards:

ASCII (American Standard Code for Information Interchange) for the United States.

ISO 8859-1 for Western European Language.

KOI-8 for Russian.

GB18030 and BIG-5 for chinese, and so on.

Problem: This caused two problems:

A particular code value corresponds to different letters in the various language standards.

The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte.

Solution:

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 bytes, so java also uses 2 bytes for characters.
Lowest Value: \u0000
Highest Value: \uFFFF

Java ASCII Values

In Java, ASCII values can be represented using integer literals. ASCII stands for American Standard Code for Information Interchange. ASCII is a standard data-transmission code that is used by the computer for representing both the textual data and control characters. ASCII having 128 characters, i.e., from 0 to 127.

- ASCII represents a numeric value for each character, such as 65 is a value of A. In Java, an ASCII table is a table that defines ASCII values for each character.
- Currently defines 95 printable characters now defined by it, including 26 upper cases (A to Z), 26 lower cases, 10 numbers (0 to 9), and 33 special characters such as mathematical symbols, punctuation marks, and space characters.
- ASCII printable characters (33-126).
- We can convert a character to its corresponding ASCII value by casting the character to an integer using (int). Similarly, we can convert an ASCII value to its corresponding character by casting the integer to a character using (char).

Character	Character Name	ASCII code	Character	Character Name	ASCII code	Character	Character Name	ASCII code
!	Exclamation point	33	**A**	Uppercase a	65	**a**	Lowercase a	97
“	Double quotation	34	**B**	Uppercase b	66	**b**	Lowercase b	98
#	Number sign	35	**C**	Uppercase c	67	**c**	Lowercase c	99
$	Dollar sign	36	**D**	Uppercase d	68	**d**	Lowercase d	100
%	Percent sign	37	**E**	Uppercase e	69	**e**	Lowercase e	101
&	ampersand	38	**F**	Uppercase f	70	**f**	Lowercase f	102
‘	apostrophe	39	**G**	Uppercase g	71	**g**	Lowercase g	103
(	Left parenthesis	40	**H**	Uppercase h	72	**h**	Lowercase h	104
)	Right parenthesis	41	**I**	Uppercase i	73	**i**	Lowercase i	105
*	asterisk	42	**J**	Uppercase j	74	**j**	Lowercase j	106
+	Plus sign	43	**K**	Uppercase k	75	**k**	Lowercase k	107
,	comma	44	**L**	Uppercase l	76	**l**	Lowercase l	108
–	hyphen	45	**M**	Uppercase m	77	**m**	Lowercase m	109
.	period	46	**N**	Uppercase n	78	**n**	Lowercase n	110
/	slash	47	**O**	Uppercase o	79	**o**	Lowercase o	111
0	zero	48	**P**	Uppercase p	80	**p**	Lowercase p	112
1	one	49	**Q**	Uppercase q	81	**q**	Lowercase q	113
2	two	50	**R**	Uppercase r	82	**r**	Lowercase r	114

3	three	51	**S**	uppercases	83	**s**	Lowercase s	115
4	four	52	**T**	Uppercase t	84	**t**	Lowercase t	116
5	five	53	**U**	Uppercase u	85	**u**	Lowercase u	117
6	six	54	**V**	Uppercase v	86	**v**	Lowercase v	118
7	seven	55	**W**	Uppercase w	87	**w**	Lowercase w	119
8	eight	56	**X**	Uppercase x	88	**x**	Lowercase x	120
9	nine	57	**Y**	Uppercase y	89	**y**	Lowercase y	121
:	colon	58	**Z**	Uppercase z	90	**z**	Lowercase z	122
;	semi-colon	59	**[**	Left square bracket	91	**{**	Left curly brace	123
<	Less-than sign	60	\	backslash	92	|	Vertical bar	124
=	Equals sign	61	**]**	Right square bracket	93	**}**	Right curly brace	125
>	Greater-than sign	62	^	caret	94	~	tilde	126
?	Question mark	63	_	underscore	95			
@	At sign	64	`	Grave accent	96			

****

Number Systems

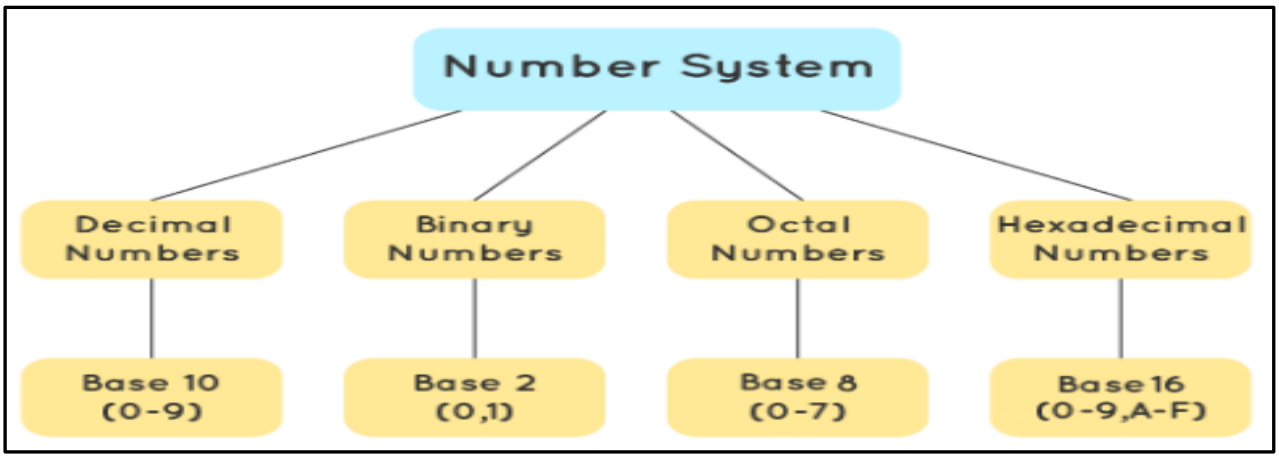
A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system.

Types of Number Systems:

There are different types of number systems in which the four main types are as follows.

1. Binary number system (Base - 2)	**2. Decimal number system (Base - 10)**
3. Octal number system (Base - 8)	**4. Hexadecimal number system (Base - 16)**

These numbers can be converted from one system to other systems like decimal to binary, decimal to hexadecimal to octal and vice versa.



1) Binary Number System:

The binary number system is a number system with base 2 in which numbers are represented only by two digits, 0 and 1. The binary number system is used commonly by computer languages like Java, C++. As the computer only understands binary language that is 0 or 1, all inputs given to a computer are decoded by it into series of 0's or 1's to process it further. These digits can be represented by off and on respectively. Binary numbers are written as 11011, 100010.

2) Decimal Number System (Base - 10):

Decimal number system is the number system we use every day and uses digits from 0 - 9 i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, & 9. The base number of the decimal number system is 10 as the total number available in this number system is 10. For example: 7310,13210,52675410 are some examples of numbers in the decimal number system.

3) Octal Number System (Base-8):

A number system with its base as eight and uses digits from 0 to 7 is called Octal Number System. The word octal is used to represent the numbers that have eight as the base. The octal numbers have many applications and importance such as it is used in computers and digital numbering systems. In the number system, octal numbers can be converted to binary numbers, binary numbers to octal numbers.

4) Hexadecimal Number System (Base - 16):

The word hexadecimal can be divided into 'Hexa' and 'deci', where 'Hexa' means 6 and 'deci' means 10. The hexadecimal number system is described as a 16-digit number representation of numbers from 0 - 9 and digits from A - F.

In the number system, hexadecimal numbers can be converted to binary numbers, binary numbers to hexadecimal numbers. For example: 7B316, 4B2A16 are hexadecimal numbers.

Conversion of Binary to Decimal Number System:

To convert a number from the binary to the decimal system, we use the following steps.

Step **1**: Multiply each digit of the given number, starting from the rightmost digit, with the exponents of the base.

Step **2**: The exponents should start with 0 and increase by 1 every time we move from right to left.

Step **3**: Simplify each of the above products and add them.

Example: Convert 100111_2 into the decimal system.

Step **1**: Identify the base of the given number. Here, the base of 100111 is 2.

Step **2**: Multiply each digit of the given number, starting from the rightmost digit, with the exponents of the base. The exponents should start with 0 and increase by 1 every time as we move from right to left. Since the base is 2 here, we multiply the digits of the given number by 20, 21, 22 and so on from right to left.

$$100111 = (1 \times 25) + (0 \times 24) + (0 \times 23) + (1 \times 22) + (1 \times 21) + (1 \times 20)$$
$$= (1 \times 32) + (0 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1)$$
$$= 32 + 0 + 0 + 4 + 2 + 1$$
$$= 39$$

Thus, $1001112 = 3910$.

Conversion of Decimal to Binary Number System: To convert numbers from decimal to binary, the given decimal number is divided repeatedly by 2 and the remainders are noted down till we get 0 as the final quotient. we use the following steps.

Step **1**: Divide the given decimal number by 2 and note down the remainder.

Step **2**: Now, divide the obtained quotient by 2, and note the remainder again.

Step **3**: Repeat the above steps until you get 0 as the quotient.

Step **4**: Now, write the remainders in such a way that the last remainder is written first, followed by the rest in the reverse order.

Example: Convert the decimal number 1310 to binary.

Step **1**: Divide 39 by 2.

Dividend	Remainder
39/2 = 19	1
19/2 = 9	1
9/2 = 4	1
4/2 = 2	0
2/2 = 1	0
1/2 = 0	1

Step **2**: Write the remainder from bottom to top i.e. in the reverse chronological order. Therefore, the binary equivalent of decimal number 39 is 100111.

****

Operators in Java

Operators are symbols that perform operations on variables and values. Consider the expression 2 + 3 = 5, here 2 and 3 are operands and + is called operator. In mathematics and computer programming, an operator is a character that represents a specific mathematical or logical action or process. There are various types of operators, each a basic symbol facilitating performing logical and mathematical processes. Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result. For **Example**: +, -, *, / etc.

There are mainly **8** Operators in Java.

1) Arithmetic Operators	**2) Relational Operators**	**3) Logical Operators**	**4) Assignment Operators**
5) Unary Operators	**6) Ternary Operators**	**7) Bitwise Operators**	**8) Shift Operators**

1) Arithmetic Operators:

Arithmetic operators are used to perform simple arithmetic(mathematical) operations on variables and data. For example, a + b;

Here, the + operator is used to add two variables a and b. Similarly, there are various other arithmetic operators in Java.

Addition (+)	**Subtraction (-)**	**Multiplication (*)**	**Division (/)**	**Modulo Division (%)**

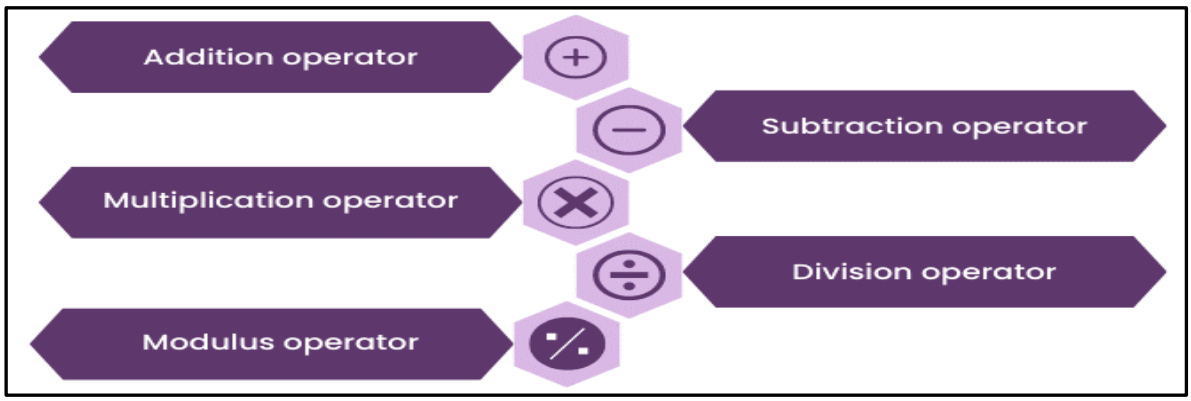
A. **Addition Operator (+):** The Addition Operator is used to add two numbers together. For instance, 'num1 + num2' will return the sum of 'num1' and 'num2'. It's the most basic Arithmetic Operation and forms the basis for many complex calculations.

B. **Subtraction Operator (-):** The Subtraction Operator subtracts one number from another. In the case of 'num1 - num2', the operation will yield the result of 'num1' minus 'num2'.

C. **Multiplication Operator (*):** The Multiplication Operator multiplies two numbers together. If you run 'num1 * num2', you'll get the product of 'num1' and 'num2'. This operation is often used in loops and array calculations.

D. **Division Operator (/):** The Division Operator divides one number by another. When you execute 'num1 / num2', it returns the quotient of 'num1' divided by 'num2'. Be cautious, though, as Java will return an integer if both numbers are integers, even when the exact answer would be a decimal.

E. **Modulus Operator (%):** The Modulus Operator gives the remainder of a division operation. If you implement 'num1 % num2', it will return the remainder when 'num1' is divided by 'num2'. This operator is extremely useful when you need to find out if a number is even or odd, among other applications.



Operator	Name	Description	Statement	Example
+	Addition	Adds together two values	x + y	2+3=5
-	Subtraction	Subtracts one value from another	x – y	5-3=2
*	Multiplication	Multiplies two values	x * y	3*4=12
/	Division	Divides one value by another	x/y	4/2=2
%	Modulus	Returns the division remainder	x % y	7%3=1

Example of a Java Program to Perform Arithmetic Operators:

```
public class ArithmeticOperations {
  public static void main(String[] args) {
    int num1 = 10;
    int num2 = 5;

    System.out.println("Addition: " + (num1 + num2));
    System.out.println("Subtraction: " + (num1 - num2));
    System.out.println("Multiplication: " + (num1 * num2));
    System.out.println("Division: " + (num1 / num2));
    System.out.println("Modulus: " + (num1 % num2));
  }
}
```

Output

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
Modulus: 0
```

When there is more than one arithmetic operator in an expression follow Precedence/ Priority rules. Above five Arithmetic operators are divided into two types of Priority.

High Priority: * / %

Low Priority: + -

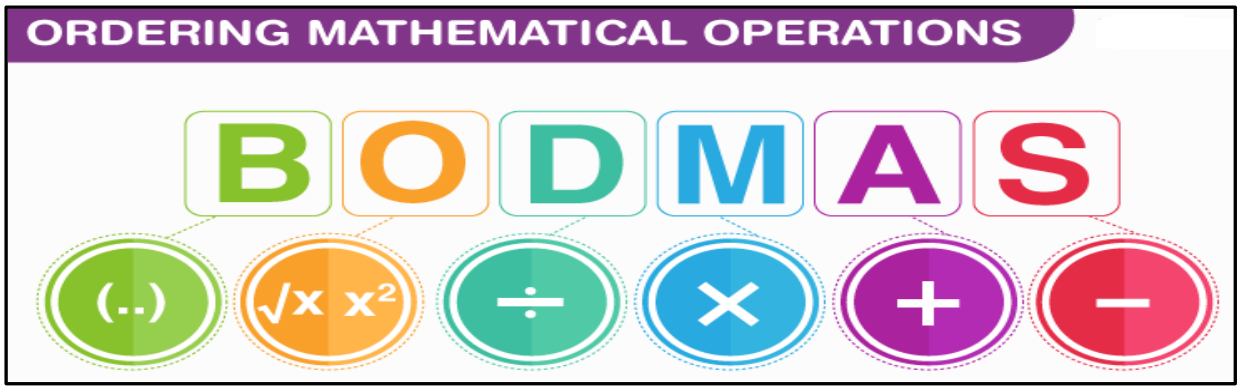
Here * / % have equal priority but when compare to + - it is having high priority
Here + - have equal priority but when compare to * / % it is having low priority
Evaluation: left to right.

Note:1 When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last.

Note:2 Arithmetic operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear.

BODMAS Rule: It stands for, B – Brackets, O – Order of powers or roots, D – Division, M – Multiplication , A – Addition and S – Subtraction.

BODMAS is a set of rules or an order to perform an arithmetic expression so that evaluation becomes easier. According to BODMAS when we have to solve an expression, we have to first solve the expression with brackets then with the exponents, division, multiplication, addition, and subtraction. If you just do without following this rule you will get the answer wrong.



Example1: (3+4)5+6-2 According to BODMAS:
The first step is to add the numerical that is in the bracket that is **3+4=7.**
The next step is to multiply 7 with **5=7x5=35.**
The next step is to add **35+6=41.**
Then subtract 2 that is **41-2=39.**

Example2,3:

$36 \div 6 \times 3 + 2^2 - (3 + 5)$

$= 36 \div 6 \times 3 + 2^2 - 8 \longleftrightarrow$ Brackets: $(3 + 5)$

$= 36 \div 6 \times 3 + 4 - 8 \longleftrightarrow$ Order of Powers: 2^2

$= 6 \times 3 + 4 - 8 \longleftrightarrow$ Division: $36 \div 6$

$= 18 + 4 - 8 \longleftrightarrow$ Multiplication: 6×3

$= 22 - 8 \longleftrightarrow$ Addition: $18 + 4$

$= 14 \longleftrightarrow$ Subtraction: $22 - 8$

Simplify the expression by using the BODMAS rule:
$[18 - 2(5 + 1)] \div 3 + 7$

Solution:

The given expression is $[18 - 2(5 + 1)] \div 3 + 7$

- **Step 1:** We begin with solving the innermost bracket first. Starting with $5 + 1 = 6$. Thus, $[18 - 2(6)] \div 3 + 7$
- **Step 2:** Next, we work with the order, thereby multiplying 2 (6) or $2 \times 6 = 12$. Thus, $[18 - 12] \div 3 + 7$
- **Step 3:** There is one bracket left, $[18 - 12] = 6$. So, $6 \div 3 + 7$
- **Step 4:** After B and O comes D, hence, $6 \div 3 = 2$. So, $2 + 7$
- **Step 5:** And finally, addition, $2 + 7 = 9$

∴ The expression is simplified and the answer is 9.

For Practice:

- ✓ **Solve:** $8 + 9 \div 9 + 5 \times 2 - 7$.
- ✓ **Solve:** $3 + 24 \times (15 \div 3)$
- ✓ **Solve:** $2 + 5(4 + 2) + 32 - (1 + 6 \times 3)$
- ✓ **Solve:** $17 - 24 \div 6 \times 4 + 8$
- ✓ **Solve:** $(9 \times 3 \div 9 + 1) \times 3$

2) Relational Operators:/Comparison Operators:

Relational operators in Java are used to establish a relationship between two operands. They compare the values and return a boolean result, either true or false. These operators are primarily used in conditions that control the flow of the program, such as in if statements, loops (for, while, do-while), and ternary operations. A relational operator compares two values and determines the relationship between them. Java Relational Operators are used to check the relation between values or variables like equality, greater than, and less than. The output of relational operators is always a boolean value. Relational operators are important in programming, because it helps us to find answers and make decisions. There are following relational operators supported by Java language.

< : Less than	**> : Greater than**	**== : Equal to**
<= : Less than or equal to	**>= : Greater than or equal to**	**! = : Not equal**

Syntax: operand1 relational_operator operand2
operand1 - The first variable to be compared.
operand2 - The second variable to be compared.
relational_operator - The relational operator to check relation between operand1 and operand2.
Ex: **a > b**

Operator	Name	Statement	Description	Example	Output
<	Less than	x<y	Checks if the left operand is less than the right operand	5<9	True
<=	Less than or equal to	x <= y	Checks if the left operand is less than or equal to the right operand	7<=2	False
>	Greater than	x>y	Checks if the left operand is greater than the right operand	6>2	True
>=	Greater than equal to	x>=y	Checks if the left operand is greater than or equal to the right operand	3>=4	False
==	Equal to	x == y	Checks if two values are equal	2==5	False
!=	Not Equal	x! = y	Checks if two values are not equal	9! =4	True

When there is more than one relational operator in an expression follow Precedence/ Priority rules.
Above 6 Relational Operators are divided into two types of Priority.
High priority: < <= > >=
Low priority: == ! =
Evaluation: left to right.
Output type of data: boolean either **true or false.**
Note: When compare to Relational operator, Arithmetic operator is having high priority.
Example:

a	b	a<b	a<=b	a>b	a>=b	a==b	a! =b
10	20	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
10	10	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE

Example Java Program to Perform Relational Operators:

```java
//Java Program to perform relational operations
public class Main
{
   public static void main(String args[])
   {
      //Declare and initialize the variables
      int a = 6;
      System.out.println("The entered value of a is " + a);
      int b = 2;
      System.out.println("The entered value of b is " + b);
      //Perform relational operations
      System.out.println("Values of Relational Operations: ");
      System.out.println("The result of a == b is " + (a == b) );
      System.out.println("The result of a != b is " + (a != b) );
      System.out.println("The result of a > b is " + (a > b) );
      System.out.println("The result of a < b is " + (a < b) );
      System.out.println("The result of b >= a is " + (b >= a) );
      System.out.println("The result of b <= a is " + (b <= a) );
   }
}
```

OUTPUT:

```
The entered value of a is 6
The entered value of b is 2
Values of Relational Operations:
The result of a == b is false
The result of a != b is true
The result of a > b is true
The result of a < b is false
The result of b >= a is false
The result of b <= a is true
```

3) Logical Operators / Conditional Operators:

Logical operators in Java are special symbols or keywords that perform logical operations on Boolean values. These operators allow developers to combine or manipulate Boolean expressions, resulting in a single Boolean value. Logical operators in Java help us combine multiple conditional statements. When applied to two conditions, these operators return logical output. These operators are used to perform "logical AND", "logical OR" and "logical NOT" operations on the two operands, the functions which are similar to AND and OR gate in digital electronics... Used extensively to test for several conditions for making a decision. You can also test for true or false values with logical operators. There are primarily three types of logical operators.

Logical Operator	Java Operator
AND	&&
OR	||
NOT	!

A. Logical AND (&&) in Java:

Java Logical AND operator, when both conditions are true the value returned is true, but even if one condition is false the value returned will be false and if both the conditions are false value returned is false. The AND operator is represented by two ampersands (&&).

Java's logical AND operator syntax is:

boolean result = condition1 && condition2;

The **Truth Table** or the results of the AND operation.

Condition1	Condition2	Condition1 && Condition2
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE

From the table above, we can see that the AND operator returns true only if both conditions are true. Even if one of the conditions is false, the AND operator returns false.

B. Logical OR (||) in Java:

This operator returns true if any one of the given conditions is true, also if both the conditions are true it will still return true. The only way of getting a false out of this operator is both the conditions being false. This operator doesn't check the second condition if the first one is true. The second condition is checked only and only if the first condition is false. The OR operator is represented by two vertical bars (||).

Java's logical OR operator syntax is:

boolean result = condition1 || condition2;

The **Truth Table** or the results of the OR operation.

Condition1	**Condition2**	**Condition1 && Condition2**
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

The above-given table clearly shows that the OR operator returns false if both conditions are false, else it returns true.

C. Logical NOT (!) in Java:

It returns the operand's logical state in reverse. The NOT logical operator reverses the true/false outcome of the expression. Finally, NOT logical operator returns true when the condition is false. The operator accepts a single value as an input and returns its inverse. The operator simply changes the value of the expression to the opposite.

Java's Logical NOT Operator Syntax: boolean result =! operand;

The **Truth Table** or the results of the negation operation.

Condition1	**! (Condition1)**
TRUE	FALSE
FALSE	TRUE

If the condition is true, the operator returns false, which is the opposite of true.

Logical Operators Truth Table:

Value1	**Value2**	**&&** (Value1 &&Value2)	**||** (Value1 || Value2)	**!** ! (Value1)
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	TRUE

Operator	**Name**	**Description**	**Example** (x=3, y=5)	**Result**
&&	Logical AND	Returns true when both conditions are true	x < 5 && y < 10	TRUE

||	Logical OR	Returns true if at least one condition is true	x < 5 || y < 4	TRUE
!	Logical NOT	Reverse the result, returns false if the result is true	! (x < 5 && y < 10)	FALSE

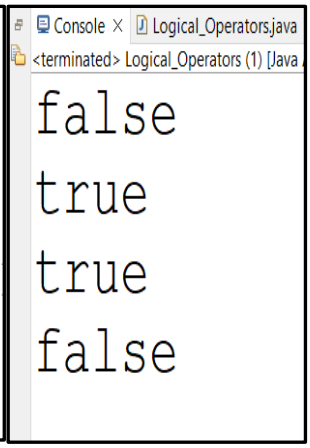
Example Java Program to Perform Logical Operators:

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

//boolean value1=true,value2=true;
boolean value1=true,value2=false;
//boolean value1=false,value2=true;
//boolean value1=false,value2=false;

System.out.println(value1&&value2); //
System.out.println(value1||value2); //

System.out.println(!(value1&&value2));
System.out.println(!(value1||value2));
}
```



D. Assignment Operators/Compound Assignment Operators:

Java Assignment operators are used to assign values to the variables on the left side of the equal's sign. we use the assignment operator '=' to assigning a value to any variable. The associativity is from right to left i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a Compound Statement. For example, instead of a = a+5, we can write a += 5.

Following are the assignment operators supported by Java language.

+=	left = left+ right
-=	left= left- right
*=	left=left* right
/=	left=left/ right
%=	left=left% right

The value on the right side must be of the same data type of the operand on the left side. Otherwise, the compiler will raise an error. This means that the assignment operators have right to left associativity, i.e., the value given on the right-hand side of the operator is assigned to the variable on the left.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

Operator	Name	Statement	Example	Result
=	Simple assignment operator	x = 5	x=5	5
+=	Add AND assignment operator	x += 3	x = x + 3	x=8
-=	Subtract AND assignment operator	x -= 3	x = x - 3	x=2
*=	Multiply AND assignment operator	x *= 3	x = x * 3	x=15
/=	Divide AND assignment operator	x/=3	x = x / 3	x=1
%=	Modulus AND assignment operator	x %= 3	x = x % 3	x=2

5) Unary Operators:

Unary operators require only one operand to perform any operations like increment, decrement, negation, etc. It consists of various operators that support working on a single operand. These operations can include incrementing or decrementing a value, reversing a boolean value, or other types of calculations. Examples of unary operators include the increment operator (++), decrement operator (--), and negation operator (!). Unary operators are operators that act upon a single operand to produce a new value. An operator that acts on a single operand is called unary operator. It uses a single variable.

Types of Unary Operators in Java:

Symbol	Operator Name	Description	Syntax	Example (x=5)
+	Unary Plus	It is used to denote a positive value.	+x	+5
-	Unary Minus	It is used to denote a negative value.	-x	-5
++	Increment Operator	Increments value by 1	x++ (x+1)	5+1=6
- -	Decrement Operator	Decrements value by 1	x-- (x-1)	5-1=4
!	Logical Complement Operator	Reverse the value of boolean variable	! x (x=true)	false

A. Unary Plus Operator:

Unary Plus is used to represent positive values. In this, the operator is optional; that is, we do not have to write the operator before the operand. It is used to represent a positive operand. The unary plus operator in Java is represented by the symbol +. The type of the operand must be an integer data type, or a compile-time error occurs.

B. Unary Minus Operator:

The unary minus operator (-) can appear as part of a unary expression. This operator can be used to convert a positive value to a negative one. The unary minus operator negates a numeric value, changing its sign. This operator is used on numbers, it changes the positive number to negative number. However, using it with an already negative value will convert it into a negative number. The type of the operand of the unary - operator must be an arithmetic data type, or a compile-time error occurs.

C. Increment Operator (++):

The increment operator in Java increases the value of the variable(operand) by one (1). The symbol for the operator is two plus signs (++). The operator can be applied before or after the operand. Both will have the same increment of 1. This operator is used to increment the existing value of the operand.

There are **Two** forms of the increment operator:

Pre-Increment Operator (++a): Pre-increment Operator is used before the variable name (++a), the operand's value is incremented instantly. The Pre-increment operator works by incrementing the value by one (1) before returning the new incremented value. In the pre increment the value is incremented at first, then used inside the expression. When the increment is placed before the operator. This means the value is incremented instantly. This operator will not wait for the execution of the statement, and it automatically increments the value.

If the expression is a = ++b; and b is holding 5 at first, then a will hold 6. Because increase b by 1, then set the value of a with it.

Post-Increment Operator (a++): Post-increment operator is placed after the variable name, the operand's value is increased, but the previous value is maintained temporarily until the end of this statement and updated before the end of the next statement. The post-increment operator works by initially returning the value of the operand and finally incrementing it by one (1). The post increment operator is used to increment the value of some variable after using it in an expression. In the post increment the value is used inside the expression, then incremented by one. When the increment is placed after the operator. This means the value is not incremented instantly. The value of the operand is incremented, but the previous value is retained temporarily until the execution of this statement, and it gets updated before the execution of the next statement.

If the expression is a = b++; and b is holding 5 at first, then a will also hold 5. Because increase b by 1 after assigning it into a.

D. Decrement Operator (--):

Decrement as the name implies is used to reduce the value of a variable by 1. It is also one of the unary operator types, so it can be used with a single operand. Java decrement operator is just as opposite to the increment operator. It decreases the value of the variable by 1. The symbol for the operator is two minus signs (--). Same as of increment operator it can also be applied before and after an operand.

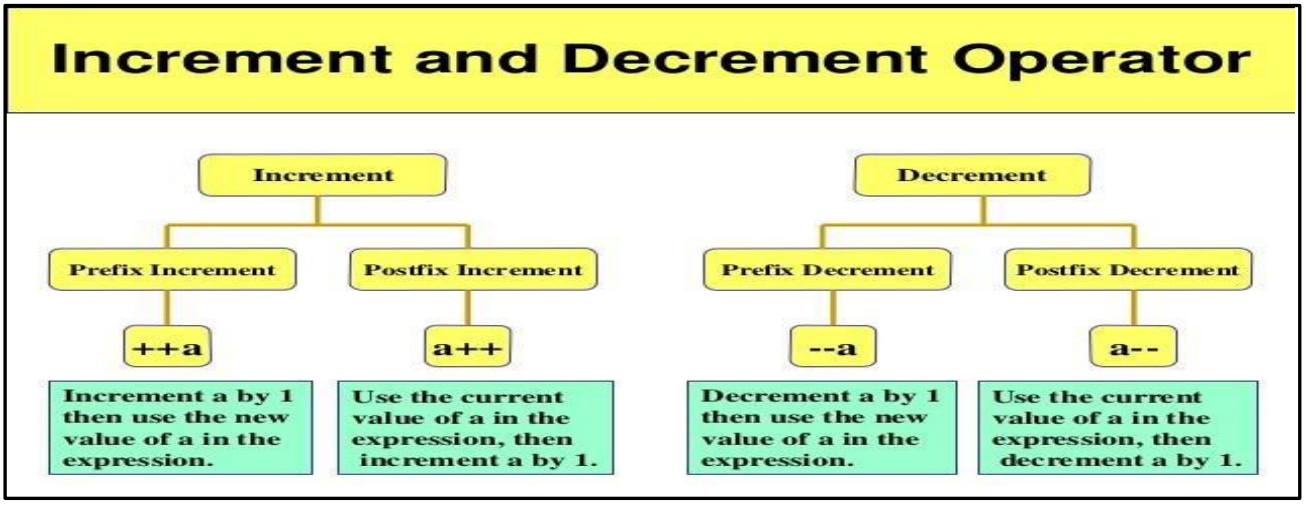
Pre-Decrement Operator (--a): The pre-decrement operator is defined as when the decrement operator (--) is written before a variable; then, it is known as the Pre-decrement operator or Prefix. In the prefix form, the decrement unary operators appear before the operand. While using the prefix form, we first update the value of the operand and then we use the new value in the expression. When the decrement is placed before the operator. This means the value is decremented instantly. This operator will not wait for the execution of the statement, and it automatically decrements the value.

If the expression is a = --b; and b is holding 5 at first, then a will hold 4. Because decrease b by 1, then set the value of a with it.

Post-Decrement Operator (a--): Post-decrement operators, also known as Postfixes, are created when the decrement operator (--) is written after a variable. In the postfix form, the operator appears after the operand. While using the postfix form, we first use the value of the operand in the expression and then update it.

When the decrement is placed after the operator. This means the value is not decremented instantly. The value of the operand is decremented, but the previous value is retained temporarily until the execution of this statement, and it gets updated before the execution of the next statement.

If the expression is a = b--; and b is holding 5 at first, then a will also hold 5. Because decrease b by 1 after assigning it into a.



Difference between Pre-Increment and Post-Increment, Pre-Decrement and Post-Decrement in Java:

Pre-Increment	Post-Increment	Pre-Decrement	Post-Decrement
Increments the value of variable by 1 before using it in the expression.	Uses the current value of variable in the expression, then increments it by 1.	Decrements the value of variable by 1 before using it in the expression.	Uses the current value of variable in the expression, then decrements it by 1.
Pre-increment occurs before any other operation in the expression.	Post-increment occurs after all other operations in the expression.	Pre-decrement occurs before any other operation in the expression.	Post-decrement occurs after all other operations in the expression.
Useful when you need the updated value of the variable immediately.	Useful when you need to use the original value of the variable in the expression.	Useful when you need the updated value of the variable immediately.	Useful when you need to use the original value of the variable in the expression.
Returns the updated value of the variable.	Returns the original value of the variable.	Returns the updated value of the variable.	Returns the original value of the variable.
Works with variables of numeric types such as int, long, float, and double.	Works with variables of numeric types such as int, long, float, and double.	Works with variables of numeric types such as int, long, float, and double.	Works with variables of numeric types such as int, long, float, and double.
++variable	variable++	--variable	variable--
int x = 5; int y = ++x; // Now x is 6, y is 6	int a = 5; int b = a++; // Now a is 6, b is 5	int x = 5; int y = --x; // Now x is 4, y is 4	int a = 5; int b = a--; // Now a is 4, b is 5

E. Logical Not Operator (!):

The not operator is a logical operator, represented in Java by the ! symbol. It's a unary operator that takes a boolean value as its operand. Java logical Not operator flips the value of a boolean value. A logical Complement Operator is used to reverse the value of a Boolean value. It means that if the value of an operand is true, then the complement of the operator will be false and vice-versa. It is represented by an exclamatory symbol (!).

"If value of a boolean variable 'b' is true, then !b is false. If the value of 'b' is false, then !b is true."

6) Ternary Operators:

Ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name ternary. It consists of a condition followed by a question mark(?). It contains two expressions separated by a colon (:). If the condition evaluates to true, then the first expression is executed, else the second expression is executed. It is also referred to as a miscellaneous operator. We can use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators. Although it follows the same algorithm as of if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

Syntax: Variable = Condition ? Expression1 : Expression2 ;

Condition: It represents the condition that is written in an if statement.

Expression1: This expression will be stored in the Variable if the condition turns out to be true.

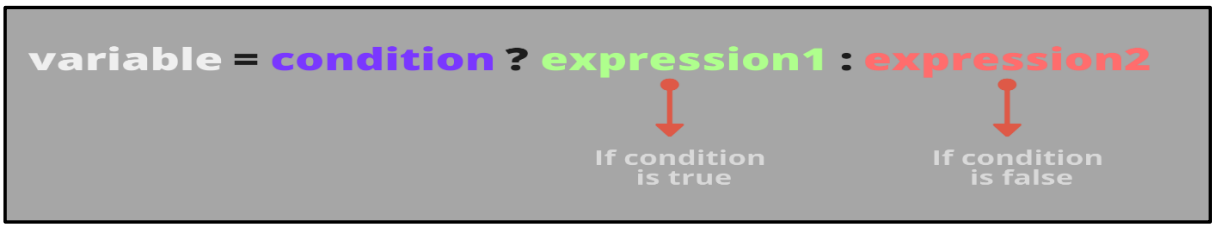
Expression2: This expression will be stored in the Variable if the condition turns out to be false.

Variable: It stores the value returned by either expression.

Ex: (marks > 40) ? "pass" : "fail";

If marks=70, then the first condition is true, so the first statement is executed o/p is pass.

If marks=30, the second condition is true, so the second statement is executed o/p is fail.



7) Bitwise Operators:

Java Bitwise operators are generally used to perform operations on bits of data. We can use bitwise operators with any integral datatype like byte, short, int, long. The individual bits of a number are considered in calculation and not the entire number itself.

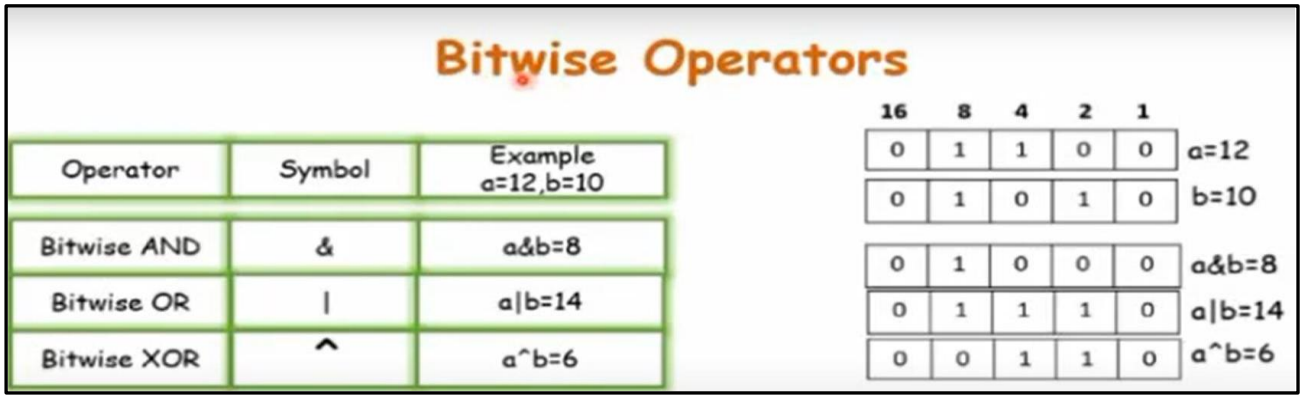
There are the following types of **Bitwise Operators**:

Bitwise AND (&)	Bitwise OR (|)	Bitwise XOR (^)	Bitwise COMPLEMENT (~)

A. Bitwise AND (&) Operator: This is a binary operator, which is denoted by symbol '&'. It performs bit by bit AND operation on given number, if both bits are 1, then output will be 1, otherwise it will 0.

B. Bitwise OR (|) Operator: This is a binary operator, which is denoted by symbol '|'. It performs bit by bit OR operation on given number, if any bit is 1, then output will be 1, otherwise it will 0.

C. Bitwise XOR Operator (^): This is a binary operator, which is denoted by symbol '^'. It performs bit by bit XOR operation on given number, if both bits are different, then output will be 1, otherwise it will 0.



D. Bitwise COMPLEMENT (~) Operator:

The bitwise complement operator is a unary operator (works with only one operand). It is denoted by '~'. It performs bit by bit 1's complement operation on given number. It inverted the bits. There is a simple way to find the 1's complement of any binary numbers. It is formed by changing 1's into 0's and 0's into 1's. It changes binary digits 1 to 0 and 0 to 1.

It is important to note that the bitwise complement of any integer **N is equal to - (N + 1).**

For example:

Consider an integer 35. As per the rule, the bitwise complement of 35 should be -(35 + 1) = -36. Now let's see if we get the correct answer or not.

35 = 00100011 (In Binary

(~) 00100011) // using bitwise complement operator

11011100

In the above example, we get that the bitwise complement of 00100011 (35) is 11011100. Here, if we convert the result into decimal, we get 220. However, it is important to note that **we cannot directly convert the result into decimal and get the desired output**. This is because the binary result 11011100 is also equivalent to -36.

To understand this, we first need to calculate the binary output of -36.

2's Complement:

In binary arithmetic, we can calculate the binary negative of an integer using 2's complement. 1's complement changes 0 to 1 and 1 to 0. And, if we add 1 to the result of the 1's complement, we get the 2's complement of the original number.

For example, // compute the 2's complement of 36

36 = 00100100 (In Binary)

1's complement = 11011011

2's complement:

11011011

+ 1

11011100

Here, we can see the 2's complement of 36 (i.e. -36) is 11011100. This value is equivalent to the bitwise complement of 35.

Hence, we can say that the bitwise complement of 35 is -(35 + 1) = -36.

1's complement and 2's complement:

Generally, there are two types of complement of Binary number: 1's complement and 2's complement. To get 1's complement of a binary number, simply invert the given number. For example, 1's complement of binary number 110010 is 001101. To get 2's complement of binary number is 1's complement of given number plus 1 to the least significant bit (LSB).

For example: 2's complement of binary number 10010 is (01101) + 1 = 01110.

There is a simple algorithm to convert a binary number into 1's complement. To get 1's complement of a binary number, simply invert the given number.

Example: Find 1's complement of binary number 10101110.

Simply invert each bit of given binary number, so 1's complement of given number will be 01010001.

2's Complement of Unsigned Binary Numbers:

To find the 2's complement of the binary number first we have to find the 1's complement of this binary number and then add 1 to the left most bit of the binary number. The 2's complement of a binary number is formed by changing 1's into 0's and 0's into 1's or taking the 1's complement and then adding 1 to the least significant bit.

For example:

Find the 2's complement of 1011001?

Solution: First, we will change all 1's into 0's and all 0's into 1's. So, by applying this rule, 1011001 will become 0100110. Now, we will add 1 to the least significant bit. Thus 0100110 + 1 = 0100111. Therefore, the required answer is 0100111.

[Signed integers are stored in a computer using 2's complement. It consists both negative and positive values but in different formats like (-1 to -128) or (0 to +127). An unsigned integer can hold a larger positive value, and no negative value like (0 to 255)]

8) Shift Operators:

In Java, shift operators are the special type of operators that work on the bits of the data. These operators are used to shift the bits of the numbers from left to right or right to left depending on the type of shift operator used. Shift operators are used to perform bit manipulation. These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two.

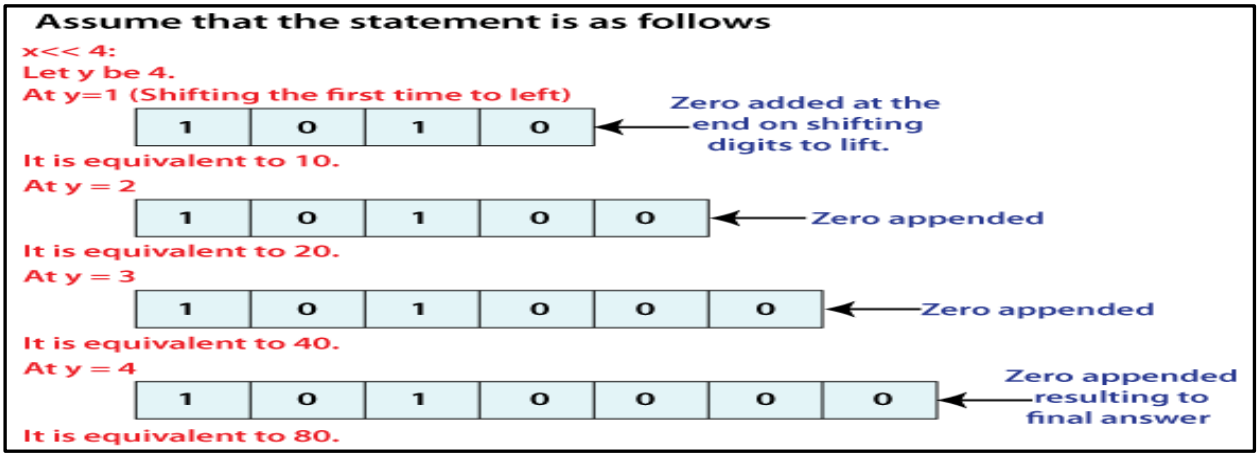
Types of Shift Operators in Java:

1.Signed Left Shift Operator (<<)	**2.Signed Right Shift Operator (>>)**	**3.Unsigned Right Shift Operator (>>>)**
The left shift operator moves all bits by a given number of bits to the left.	The right shift operator moves all bits by a given number of bits to the right.	It is the same as the signed right shift, but the vacant leftmost position is filled with 0 instead of the sign bit.

Note: Java does not support the Unsigned Left Shift Operator (<<<).

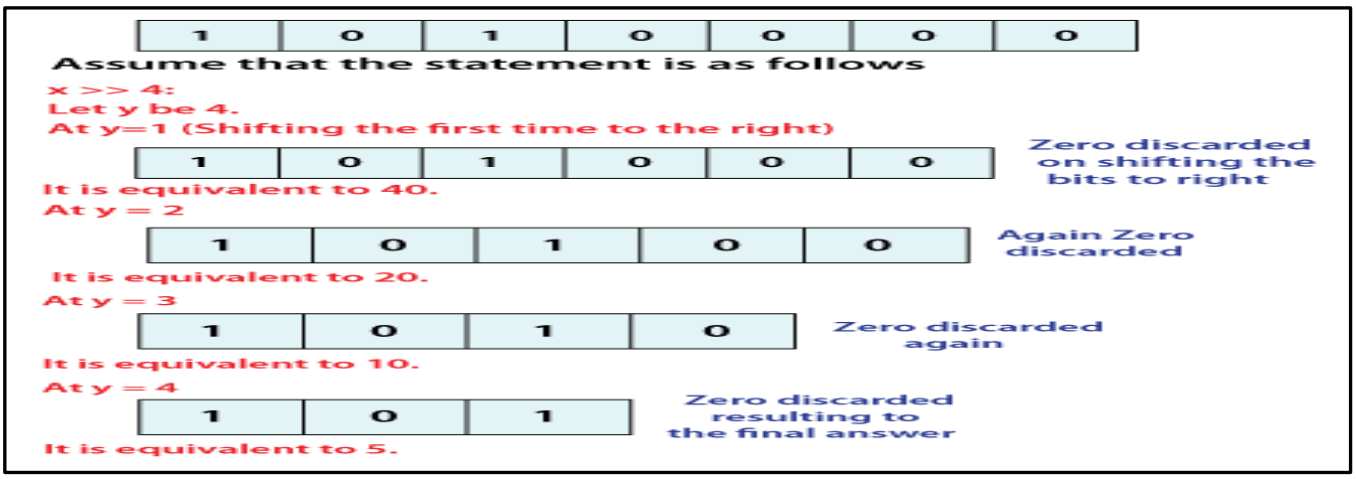
A. **Signed Left Shift Operator (<<):** The signed left shift operator is a special type of operator used to move the bits of the expression to the left according to the number specified after the operator. Left shift operator is denoted by << symbol. It shifts all bits towards left by a certain number of specified bits, **For Example**: num<<2 will shift the bits of number to the left by two positions. The bit positions that have been vacated by the left shift operator are filled with 0's. The operator that shifts the bits of number towards left by n number of bit positions is called left shift.

Example: Consider x =5, x<<4



B. Signed Right Shift Operator (>>): The signed right shift operator is a special type of operator used to move the bits of the expression to the right according to the number specified after the operator. Right shift operator is denoted by >> symbol. It shifts all bits towards right by certain number of specified bits. **For Example**: num>>2 will shift the bits to the right by two positions. The bit positions that have been vacated by the right shift operator are filled with 0's. The operator that shifts the bits of number towards the right by n number of bit positions is called right shift operator in Java.

Example: Consider x =80, x>>4



C. **Unsigned Right Shift Operator (>>>):** Unsigned Right Shift Operator moves the bits of the integer a given number of places to the right. The sign bit was filled with 0s. The Bitwise Zero Fill Right Shift Operator is represented by the symbol >>>. The unsigned right shift operator is a special type of right shift operator that does not use the signal bit to fill in the sequence. The unsigned sign shift operator on the right always fills the sequence by 0.

Operator Categories	Operator Symbol
1) Arithmetic Operators	+ ,- ,* , / , %
2) Relation Operators	<, > , <=,>=,== ,!=
3) Logical Operators	&&, ||, !
4) Assignment Operators	+=, -= , *= , /=,%=
5) Unary Operators	Unary + , - , ++ , --
6) Ternary Operators	? :
7) Bitwise operators	Bitwise AND &, Bitwise OR |, Bitwise XOR ^, Bitwise Complement ~
8) Shift Operators	<< , >>, >>>

Java Operator Precedence: Precedence of operators defines which operator will be executed first if both operators are mentioned in a single expression. This affects how an expression is evaluated. Certain

operators have higher precedence than others.

Arithmetic operator precedence:

When there is more than one arithmetic operator in an expression; multiplication, division, and modulo are calculated first (*,/,%), followed by subtraction and addition(-,+). If all arithmetic operators in an expression have the same level of precedence, the order of execution is left to right.

For **Example**, the multiplication operator has a higher precedence than the addition operator. For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Ex: a = 2; b = 5; c = 6; d = 3;

=> (a + b) * c / d; // (2+5) * 6 / 3 =>10*6/3=>10*2=>20;

For **Example**, in the expression 36 - (3 * (2 + 6 - 4)), the Install Script compiler first performs the operation 2 + 6, which yields 8, then subtracts 4 from 8, yielding 4, then multiplies 3 by 4, yielding 12, and finally subtracts 12 from 36, yielding 24.

****

Control Flow Statements in Java

Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

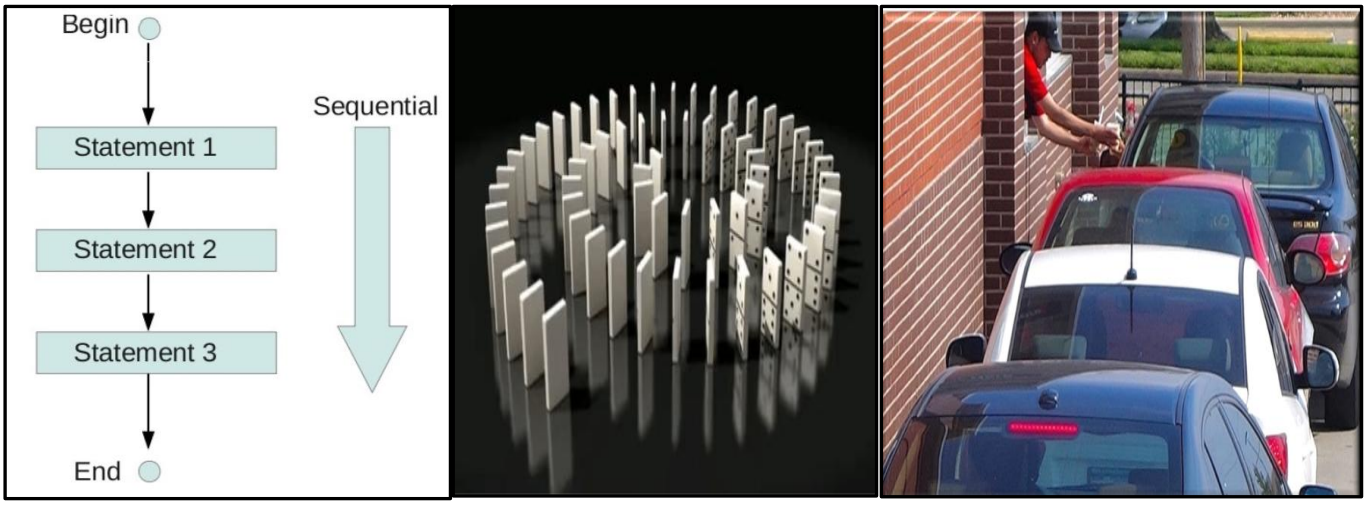
Sequential Programming in Java:

Step by Step execution of program is called Sequential Programming. Here each and every statement is important order of execution will be linear one by one no statements will be skipped. Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. A sequence is an ordered list of something. Sequential execution means that each command in a

program script executes in the order in which it is listed in the program. The first command in the sequence executes first and when it is complete, the second command executes, and so on. Sequential programming is a form of computing that executes the same sequence of instructions & always produces the same results.

Sequential programs have two main Characteristics:

- The textual order of statements specifies their order of execution.
- Successive statements must execute without any temporal overlap visible to programs.



However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Control Flow Statements in Java:

Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements in Java are instructions that manage the flow of execution of a program based on certain conditions. They are used to make decisions, to loop through blocks of code multiple times, and to jump to a different part of the code based on certain conditions. Control statements are fundamental to any programming language, including Java, as they enable the creation of dynamic and responsive programs. Control Flow statements in programming control the order in which the computer executes statements in a file. They allow us to regulate the flow of our program's execution.

In **Java**, the compiler parses the application code sequentially from top to bottom. The code statements are executed according to the order in which they appear. It's a very difficult task to model real-world problems in the code if we have no control statements. Control statements in Java are one of the fundamental features of Java which provides a smooth flow of the program. These statements decide whether a specific part of the code will be executed or not. So, it is used to control the flow of the program by executing the code on a conditional basis. In Java programming language, you can control the flow of execution of the code by placing the decision making, branching, looping, and adding conditional blocks.

There are **three main types** of flow of execution (control) that occur in any computer programming.

Sequential flow: Statements execute from top to bottom one by one.

Conditional or Selection: Out of two instructions, only one will be executed successfully based on the specified condition. This is because the condition generates the result as either true or false.

Repetition or Loop: Group of statements repeat whenever the specified condition is true.

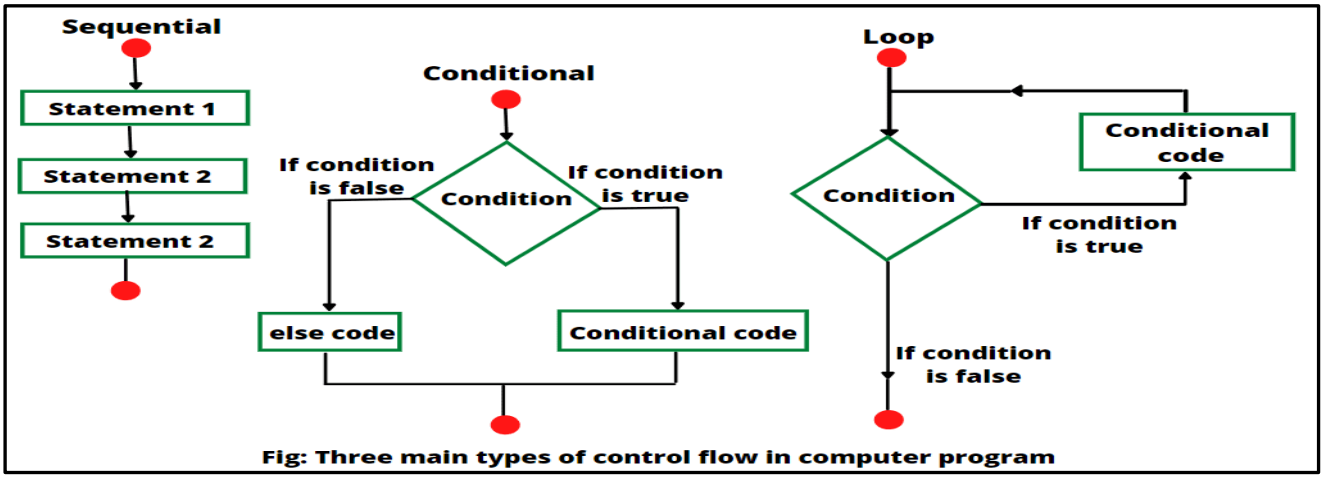


Fig: Three main types of control flow in computer program

There are **Three** Types of Control Flow Statements in Java:

1) **Decision-Making (Branching or Conditional) Statements.**
2) **Looping (Iterative or Repetition) Statements.**
3) **Jumping Statements.**

Control Flow Statements in Java are those statements that change the flow of execution and provide better control to the programmer on the flow of execution in the program.

These statements are executed randomly and repeatedly. Control statements in Java programming are used to write better and complex programs.

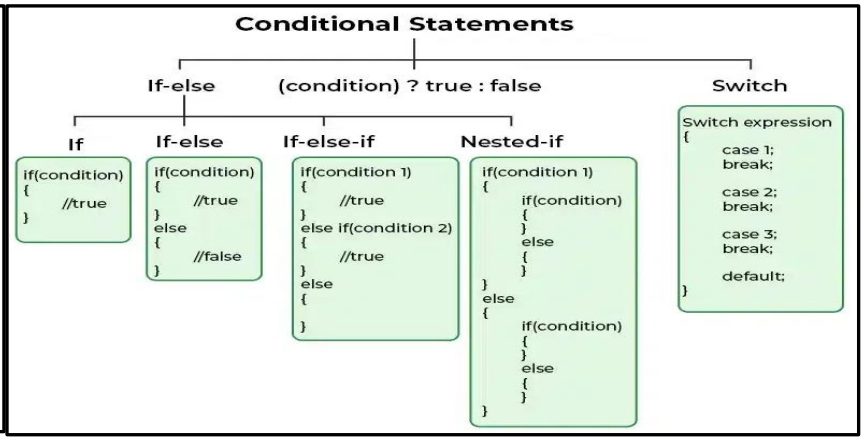
For example, suppose a situation comes in the program, where we need to change the order of execution of statements based on specific conditions or repeat a group of statements until particular specified conditions are met. Such situations can be achieved with the help of control flow statements or simply called conditional control statements in Java. Control statements are also called control structures or control constructs.

1) Decision-Making Statements [Conditional]**:**

Decision-making statements determine the program's arc and flow. Because they provide conditions using boolean expressions that are evaluated to a true or false boolean value, they are sometimes referred to as conditional statements. A particular piece of code will run if the condition is true; if the state is false, the block will not run. We generally use decision-making statements when we have to decide which block of the code will be executed first. Java decision-making statements allow you to make a decision, based upon the result of a condition. A program that breaks the sequential flow and jumps to another part of the code is called branching statements in java.

Types of Conditional Statements:

If Condition	Switch Condition
Simple-if Statement	
if-else Statement	
if-else-if Ladder	
Nested if Statement	



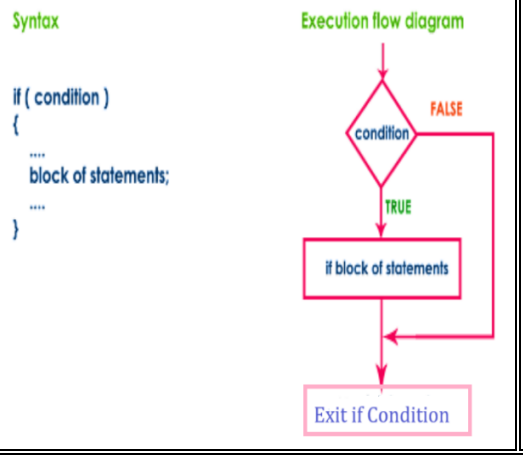
A. **If Statement:** The if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

a) **Simple If Statement:** The if statement is the most basic and direct expression for making a choice. It is used to decide whether or not to run a specific code section. A block of statements is only performed if a particular condition is met; otherwise, it is not. If statements execute a set of actions depending on the condition. If the condition evaluates to true then the code mentioned in the If block will be executed. If condition is true control move to inside body and how many statements are there it will execute, if condition is false then complete body part is skipped. Simply, "if" statement tests a condition and, if it is true, executes a block of code.

Syntax:

```
if(condition)
{
Body of if condition;
//Executes when condition
is true
Statements;
//TBS
}
```

Flowchart:



Flowchart:

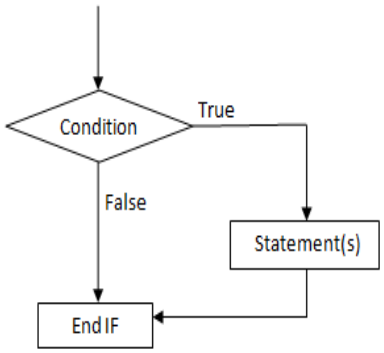


fig: Flowchart for if statement

If the condition is true, the block of code inside the curly braces is executed. If the condition is false, the block of code is skipped.

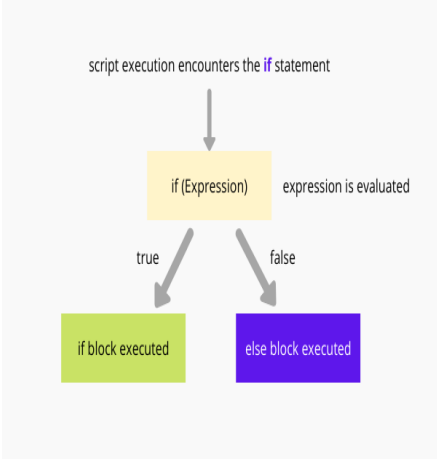
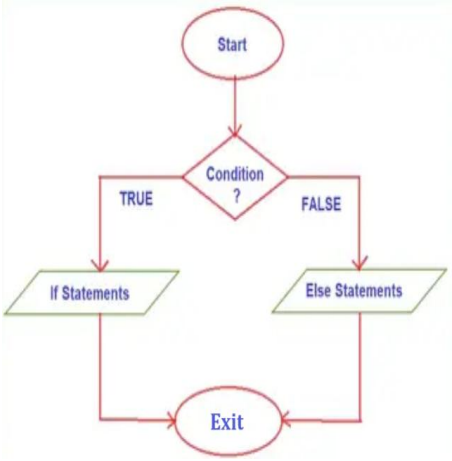
Explanation:

- STEP 1: When the program control comes to the if statement, the test expression is evaluated.
- STEP 2A: If the condition is true, the statements inside the if block are executed and exit the given condition.
- STEP 2B: If the expression is false, the statements inside the if body are not executed and exit the given condition.

b) If-Else Statement:

The **if** statement alone tells us that if a condition is true, it will execute a block of statements and if the condition is false, it won't. But what if we want to do something else when the condition is false? Here comes the C else statement. We can use the else statement with the if statement to execute a block of code when the condition is false. The if-else statement consists of two blocks, one for false expression and one for true expression.

The **if-else** statement is an extension to the if-statement by allowing an alternative set of instructions to be executed when the if condition is false, which uses another block of code, i.e. else block. If the given condition is true, then the set of actions defined under the if block will be executed and exit the if block, if the given condition is false, then the statements under the else block will be executed and exit the else block.

Syntax:	Flowchart:	Flowchart:
if(condition) { Body of if block; //Executes when condition is true Statements; //TBS } else { Body of else block; //Executes when condition is false Statements; //FBS }		

If the value of test-expression is true, then the true block of statements will be executed. If the value of test-expression if false, then the false block of statements will be executed.

Explanation: The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition (test expression).

- STEP 1: When the program control comes to the if statement, the test expression is evaluated.
- STEP 2A: If the condition is true, the statements inside the if block are executed and exit the given Condition.
- STEP 2B: If the expression is false, the statements inside the else block are executed and exit the given Condition.

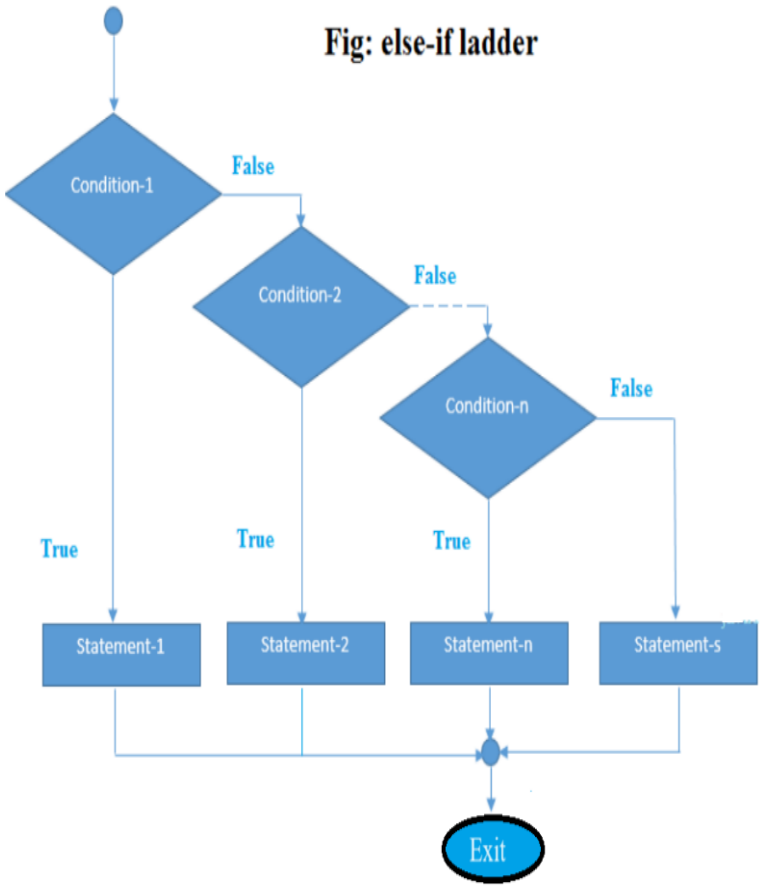
c) Else-if (or) Ladder-if (or) if-else-if ladder:

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. If else if ladder in Java programming is used to test a series of conditions sequentially. The if else if statements are used when the user has to decide among multiple options. if-else-if ladder is similar to the switch statement.

Syntax:

```
if(condition-1)
{
statement – 1;
//Code to be executed if condition1 is true
}
else if (condition -2)
{
statement – 2;
//Code to be executed if condition2 is true
}
else if (condition -3)
{
statement – 3;
//Code to be executed if condition3 is true
}
……..
……..
else if (condition – n)
{
statement – n;
}
else
{
Statement;
//Code to be executed if all the conditions are false
}
```

Flowchart:



First, the if(condition-1) block will be evaluated. If this is true, then the if block statements will be executed, and all the other blocks will be skipped. Suppose, if(condition-1) fails (if it is false), then only the next expression else if(condition-2) will be evaluated. In this case, condition-2 will be evaluated only if condition-1 is false. So, if condition-2 is false, then condition-3 will be evaluated like that. So, if none of the expressions are true, then the default block that is else block will be executed. The statements of the else block will be executed only if all the above expressions are false.

Explanation: In the above syntax, if condition-1 is true, then Statement-1 block will be executed and the remaining conditions will not be evaluated. If condition-1 is false, next condition-2 will check, if condition-2 is true, then Statement-2 block will be executed and this process will continue till end of the condition.

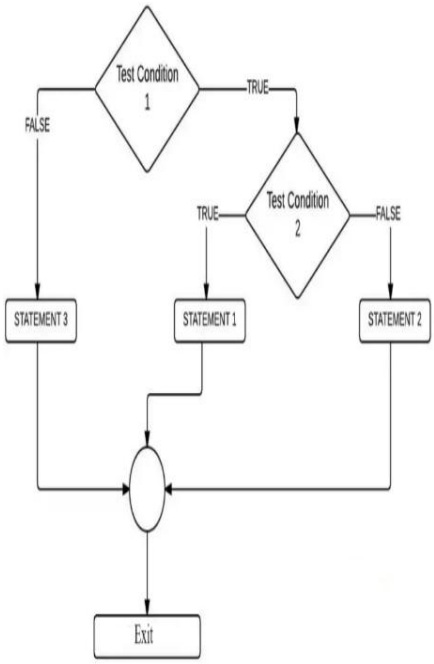
d) Nested If-Statement:

A nested if in java is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, java allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement. In some situation we want to check sequence of conditions one after other. These can be achieved by using nested- if statements. Nested means within. Nested if condition means if-within-if. If we place If Statement inside another if block is called Nested If in Java Programming. Nested-If works similar to the normal If-else condition. The only difference is that there will be an if condition inside another if condition.

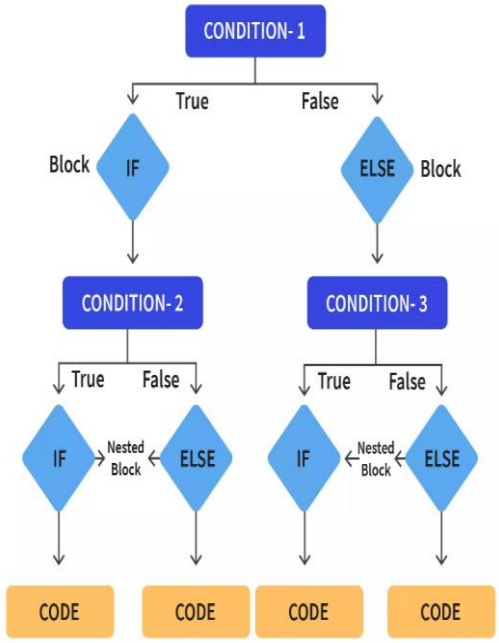
Syntax:

```
if (Condition-1)
{
//If the test Condition 1 is TRUE then these it will check for test Condition 2.
  if (Condition-2)
   {
//If the test Condition 2 is TRUE, these statements execute,
Test condition 2 True statements;
    Statement-1
  }
  else  {
 //If the c test Condition 2 is FALSE, then these statements execute,
Test condition 2 False statements;
Statements-2
  }
else  {
//If the test condition 1 is FALSE then these statements will be executed,
 Test condition 1 False statements;
Statement-3
}
```

Flowcharts:



Flowcharts:



If the Test Condition1 is FALSE, STATEMENT3 will execute. When Test Condition1 is TRUE, then it will check for the Test Condition2. If it is TRUE, STATEMENT1 will execute else STATEMENT2 will execute.

Explanation: If Condition 1 is True, then go to if condition 2. If condition 2 is true, its body will execute; otherwise, else part will execute. If Condition 1 is False, then the body of the else part will be executed. Once the condition check is finished, exit the loop. Continue the execution of statements after the loop.

- Case 1: Check the Condition-1. If the first condition is true, go for the second condition.
- Case 1.1: If Condition-2 evaluates to TRUE, the program will execute Statement-1.
- Case 1.2: Otherwise, the program will execute Statement-2.
- Case 2: If the first condition is false, go for the third one.
- Case 2.1: If Condition-3 evaluates TRUE, the program will execute Statement-3.
- Case 2.2: Otherwise, the program will execute Statement-4.

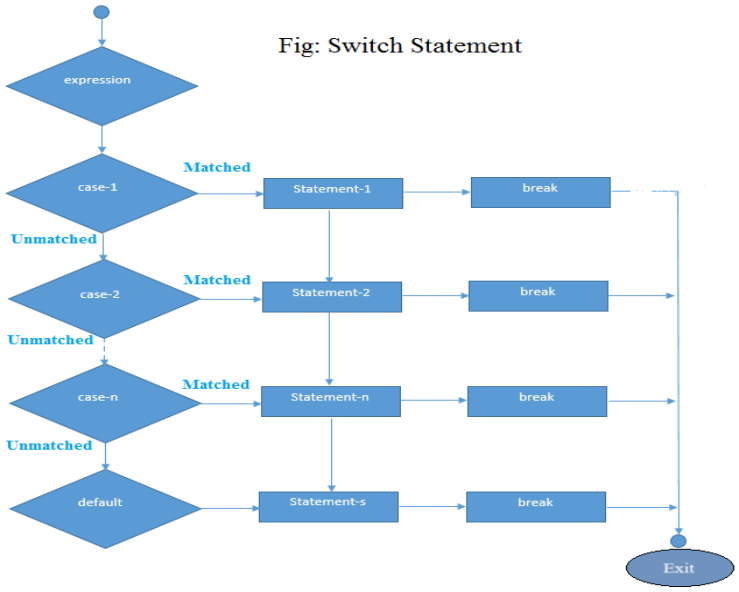
B. Switch Statement:

Writing multiple if-else statements can become quite inconvenient in complex programming scenarios where we need to make decisions based on various conditions. This is where switch case statements come in practical, providing a clean and organised way to handle multiple switch case in java with conditions. The switch statement or switch case in java is a multi-way branch statement. The switch statement selects one of many code blocks to be executed. The Java switch statement executes one statement from multiple conditions. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. Switch statements are a form of conditional statement used to Control the flow of a program. The switch-case statement is a type of decision-making statement in java. The switch statement in java is a decision control statement that is typically used when the user must choose between multiple alternatives.

Syntax:

```
switch (expression)
{
case value1 : // Code statement to be executed
statement1;
break;
//break statement is used to terminate the statement sequence
case value2 :
statement2;
break;
.
.
case value n:
statement n;
break;
default: // code to be executed if all cases are not matched;
default statement;
}
```

Flowchart:



The Switch keyword followed by an expression in parentheses. The switch expression is evaluated once. The value of the expression is compared with the values of each case. If expression matches with value1, the code of case value1 are executed. Similarly, the code of case value2 is executed if expression matches with value2.The break statement is used to terminate the statement sequence inside the switch. If there is no match, the code of the default case is executed.

There are some Rules to keep in mind while writing switch statements:

- The switch expression must be of an integer or character type.
- The case value must be an integer or character constant.
- The case value can be used only inside the switch statement.
- You can have any number of cases however there should not be any duplicates.
- The break statement in switch case is not must. It is optional.
- The break statement is used to break the flow of control once a case block is executed.
- If there is no break statement found in the case, all the cases will be executed present after the matched case.
- The default case is executed when none of the cases above match. It is optional.

Explanation: The expression inside the parentheses following the switch keyword is evaluated. The value of the expression is compared to the constant expressions in each of the case labels. If a match is found, the statements following that case label are executed. If a break statement is encountered, the switch statement is terminated, and control is transferred to the statement following the switch statement. If no break statement is encountered, control flows through to the next case label, even if a match has been found. If no match is found and the default label is present, the statements following the default label are executed. The switch statement is complete, and control is transferred to the statement following the switch statement.

Break Keyword: As java reaches a break keyword, the control breaks out of the switch block. The execution of code stops on encountering this keyword, and the case testing inside the block ends as the match is found. A lot of execution time can be saved because it ignores the rest of the code's execution when there is a break. You can use the break statement to end processing of a particular labelled statement within the switch statement. It branches to the end of the switch statement. Without break, the program continues to the next labelled statement, executing the statements until a break or the end of the statement is reached. This continuation may be desirable in some situations.

Default Keyword: The default statement is executed if no case constant-expression value is equal to the

value of expression. If there's no default statement, and no case match is found, none of the statements in the switch body get executed. There can be at most one default statement. The default statement doesn't have to come at the end. It may appear anywhere in the body of the switch statement. A case or default label can only appear inside a switch statement. The keyword is used to specify the code executed when the expression does not match any test case.

All Conditional Statements in One Table:

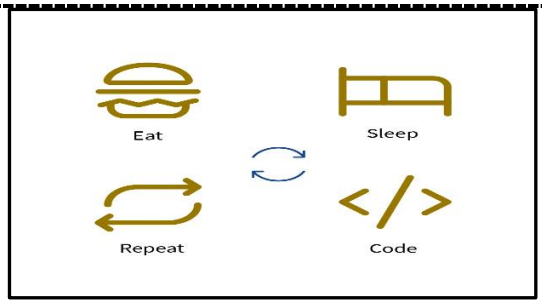
if Statement	if-else Statement	else-if Statement	Nested if Statement	Switch Statement
Only one condition is evaluated.	If the initial condition is false, the else block is executed.	If the initial condition is false, the else if condition is checked.	Conditions are checked in order until a true condition is found.	Expression is evaluated once, and control jumps to matching case.
Simple and straightforward, efficient for single conditions.	Efficient for handling two mutually exclusive conditions.	Efficient for handling multiple mutually exclusive conditions.	Efficiency decreases with increasing nesting levels.	Efficiency is comparable to if-else if chains.
Supports only one condition.	Supports two conditions: one for true, one for false.	Supports multiple conditions, each checked sequentially.	Supports multiple conditions, each within its own block.	Supports multiple cases, each associated with a value.
if (condition) { // code block }	if (condition) { // code block } else { // code block }	if (condition) { // code block } else if (condition) { // code block }	if (condition) { if (condition) { // code block } } else if (condition) { // code block }	switch (expression) { case value: // code break; case value: //code break; default: // code }
Executes the block of code if the condition evaluates to true.	Executes one block of code if the condition evaluates to true, otherwise executes another block.	Executes the block of code associated with the first true condition, ignoring the rest.	Executes the block of code associated with the first true condition encountered, possibly within nested blocks.	Executes the block of code associated with the matching case.
No support for multiple conditions in the same block.	Supports handling multiple conditions sequentially.	Supports handling multiple conditions sequentially.	Allows nesting multiple conditions within each other.	Supports handling multiple cases with unique code blocks.
Does not guarantee exclusive execution of blocks.	Guarantees exclusive execution of either if or else block.	Guarantees exclusive execution of one condition block.	Blocks can be nested to achieve exclusive execution.	Guarantees exclusive execution of one case block.
No default case is available.	The else block serves as the default case.	No explicit default case; last else if serves as default.	Can include a default case inside the nested structure.	Includes a default case to handle unmatched values.
May lead to code duplication if	Helps to avoid code duplication	Code duplication may occur if	Code duplication may occur if similar	Code duplication is minimal, as each

multiple conditions need to be checked separately.	when handling two exclusive cases.	handling similar conditions in multiple else-if blocks.	conditions are checked in nested blocks.	case is handled separately.
Suitable for simple, one-condition checks.	Suitable for handling two exclusive cases.	Suitable for handling multiple exclusive cases.	Suitable for complex, multi-condition scenarios.	Suitable for handling multiple exclusive cases based on value.

2) Looping (Iterative or Repetition) Statements:

In computer programming, loops are used to repeat a block of code. Looping statements in Java are constructs that allow you to repeatedly execute a block of code. Loops in Java are like a roller-coaster loop, or a spin cycle on a washing machine: they continue until a specific condition is met. Loops are a block of code that executes itself until the specified condition becomes false. Iteration statements or Loops in Java help to perform a task repeatedly so that we don't have to write the code explicitly again and again. That's the reason they are popularly called loops since they work in a loop to complete a particular task. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In order to understand what loops are, we have to look at some real-life cases of loops. Consider your daily routine, you wake up, you brush, you wear clothes and then head off to work, come back, eat and then sleep off. Again, the next day, you do the same things in the same order and go to sleep again. This cycle keeps on repeating. This concept of repetitive actions performed time and again is called a loop.



Need for Loops in Java:

- Loops facilitates 'Write less, Do more' - We don't have to write the same code again and again.
- They reduce the size of the Code.
- Loops make an easy flow of the control.
- They also reduce the Time Complexity and Space Complexity - Loops are faster and more feasible as compared to Recursion.

Advantages of Loops:

- Loops allow you to execute a block of code multiple times without having to rewrite it each time. This is particularly useful when you have to perform similar tasks repeatedly.
- It provides code reusability.
- Using loops, we do not need to write the same code again and again.
- By using loops, you reduce the chances of making errors in your code by eliminating repetitive tasks. This makes your code more maintainable and less prone to bugs.

There are mainly two types of loops:

A. **Entry Controlled Loops**: In this type of loops the test condition is tested before entering the loop body. An Entry-Controlled loop is a type of Loop in computer programming that tests the loop condition at the Loop's beginning before executing the Loop's body. Both **while** and **for** loops are called Entry Controlled Loops because in both of them the termination condition is checked before we enter the body of the loop hence they are called entry controlled

B. **Exit Controlled Loops:** In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is

true or false. An Exit-Controlled Loop is a type of Loop in computer programming that tests the loop condition at the end of the Loop after executing the body of the Loop at least once. Do-While loop is Exit-Controlled Loop.

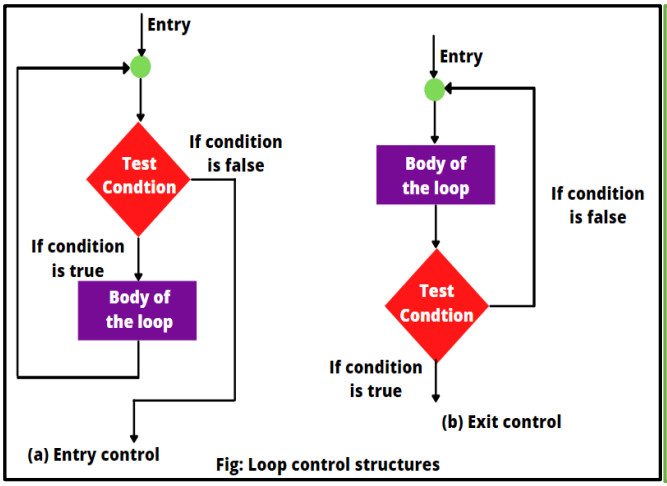
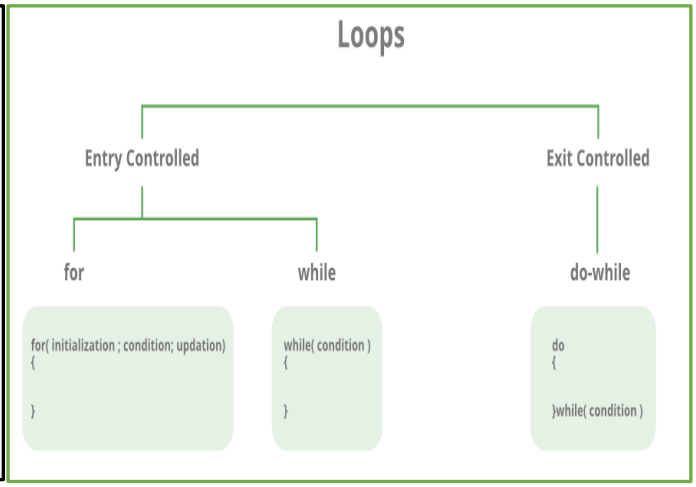
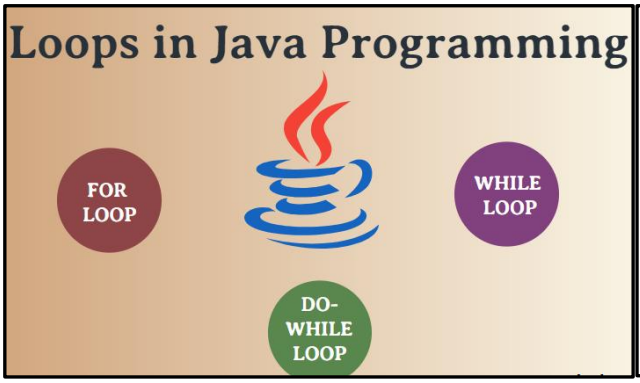


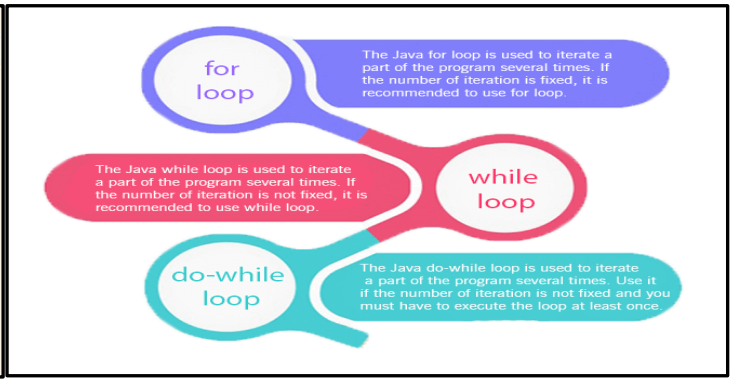
Fig: Loop control structures



In Java, we have Three Types of Loops that execute similarly. However, there are differences in their syntax and condition checking time.

a) For Loop	b) While Loop	c) Do-While Loop	d) Nested Loops





Elements of the Loops in Java:

Initialization Expression(s): Initialization is carried out only once before entering the loop. Here, we either declare and initialize a control variable(s) or only initialize the variable(s) we are going to use in looping. We can initialize multiple variables as well.

Test Expression (Condition): It is a boolean expression. Its value decides whether the loop will be executed or terminated. If the condition is satisfied, the control goes inside the loop, otherwise, it is terminated.

Body of the Loop: The statements that are to be executed repeatedly are written in the body of the Loop. They are executed until the condition of the loop is true.

Update Expression(s): Here, we update the value of the loop variable(s). It is used so that after some point of time the loop terminates. It is executed at the end of the loop when the loop-body has been executed and the next iteration is to start. It is also called Increment/Decrement expression since in most of the cases value of loop variables is either incremented or decremented.

In an exit-controlled loop, the test expression is evaluated before exiting from the loop whereas in an entry-controlled loop the test expression is evaluated before entering the loop.

a) For loop in Java:

Loops in Java are like a roller-coaster loop, or a spin cycle on a washing machine: they continue until a specific condition is met. In a for loop, that condition is a usually a numeric value. Java for loop is used to run a block of code for a certain number of times. The for loop is commonly used when you know the number of iterations in advance. Java for loop consists of 3 primary factors which define the loop itself.

These are the initialization statement, a testing condition, an increment or decrement part for incrementing/decrementing the control variable. A for loop executes a block of code as long as some condition is true. A for loop is called the counting loop because each step through the loop increases or decreases a counter each time it steps through the code. When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

```
Syntax:  for (initialization; test condition; increment/decrement)
         {
         // Body of the Loop (Statements)
         }
```

Initialization: The initialization expression is used for initializing a variable, and it is executed only once.
Condition: It executes the condition statement for every iteration. If it evaluates the condition to be true, it executes the body of the loop. The loop will continue to run until the condition becomes false.
Increment/Decrement: It is the increment or decrement statement applied to the variable to update the initial expression.
Body of the Loop: This is the code block that the loop executes for each iteration.

Flowchart:

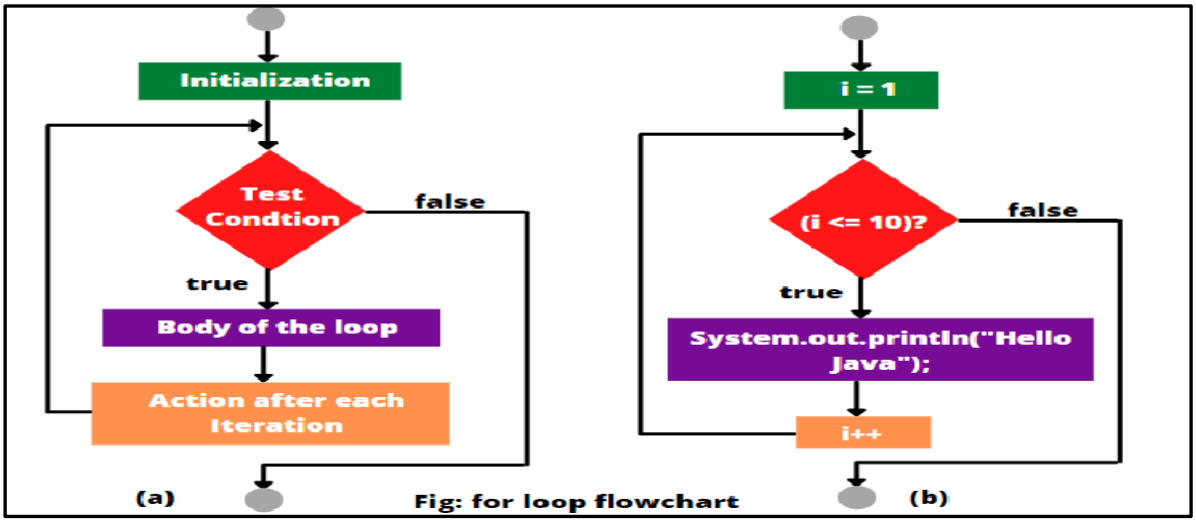


Fig: for loop flowchart

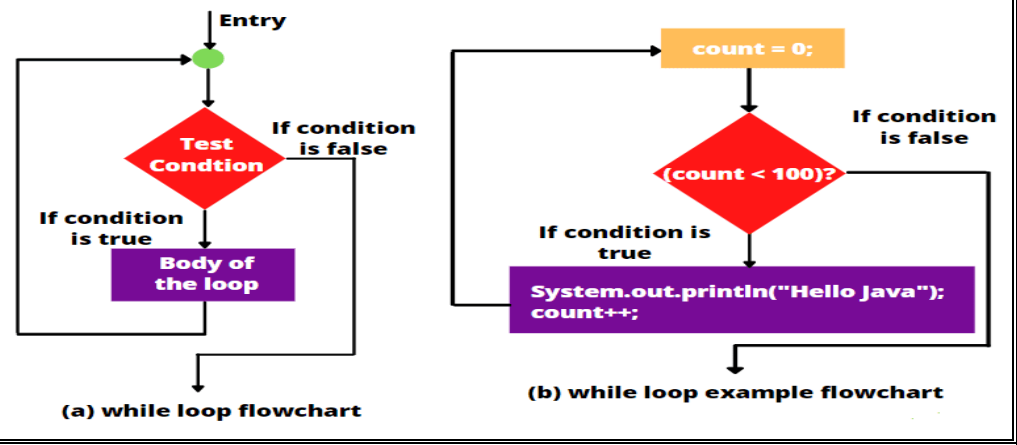
Explanation:

- The loop begins with the initialization statement (only executed once). Here, you typically declare and set your loop variable.
- Next, it checks the test condition. If the condition is true, the loop continues; if it's false, the loop ends.
- The code block inside the loop executes.
- After executing the code block, the loop hits the update statement. This usually increments or decrements your loop variable.
- The loop returns to step 2, checking the condition again. It continues this process until the test condition is false.

b) While loop in Java:

A while loop in Java is a control flow statement that executes code repeatedly based on a given Boolean expression. It executes the statements repeatedly until the Boolean expression is true, execution is terminated when the condition becomes false. The Java while loop is used to iterate a part of the program repeatedly until the specified Boolean condition is true. The while loop in Java is a pre-test loop in Java. It checks the condition before executing the loop body. As soon as the Boolean condition becomes false, the loop automatically stops. The while loop is considered as a repeating if statement. If the number of iterations is not fixed, it is recommended to use the while loop. while loop is an entry control loop. The Java while loop is used to iterate a part of the program several times. If the number of iterations is not fixed, it is

recommended to use while loop.

Syntax:	**Flowchart:**
while (test_condition) { body of loop; Statements; increment /decrement; }	

Testing Condition: Used for testing the exit condition for a loop. The expression must return a boolean value. The while loop is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.

Statement Execution: Once the condition is evaluated to true, the statements in the loop body are executed.

Increment/Decrement: Variables must be updated before the next iteration; otherwise, an infinite loop (a loop with no exit that infinitely repeats itself) could result.

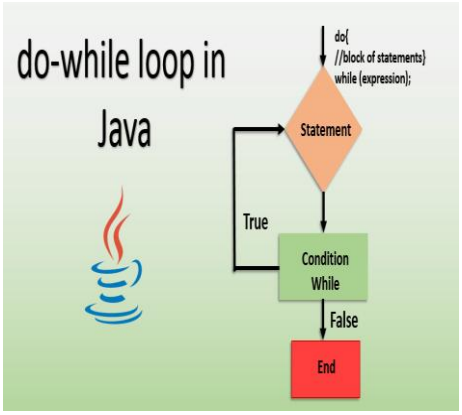
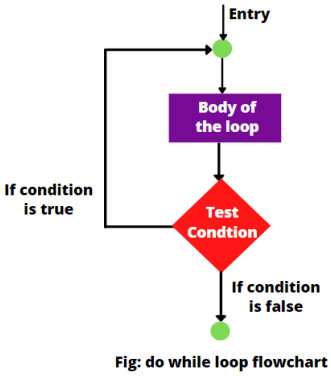
To use a Do-While loop in the Java programming, place the code block to be executed within the "do" section. After the code block, write the "while" keyword, followed by the conditional expression that must be satisfied for the loop to continue.

Explanation: A while loop executes the body of the loop as long as (or while) a Boolean condition is true. When the condition is false, we exit the loop and continue with the statements that are after the body of the while loop. If the condition is false the first time you check it, the body of the loop will not execute. The while statement looks a lot like an if statement, but it runs more than once. The curly brackets { } are optional when there is just 1 statement following the condition, but required if there are more than 1 statement in the loop. You will always use curly brackets, which is a good practice to follow.

For example: The following while loop will print 100 times "Hello Java" in a row on the screen. The variable count is the loop control variable. We continue looping as long as the value in count is less than the value in max (100). Notice that every time we go through the loop, we increment the value of count by 1. Eventually, the value in count will be larger than the value in max and we will drop out of the loop.

c) Do-While loop in Java:

The do-while loop in Java is a control flow statement that allows code to be executed repeatedly based on a given boolean condition. The primary characteristic of the do-while loop is that it guarantees the code inside the loop will execute at least once. This is because the condition is evaluated after the body of the loop has been executed. Java do-while loop is used to execute a block of statements continuously until the given condition is true. The do-while loop in Java is similar to while loop except that the condition is checked after the statements are executed, so do while loop guarantees the loop execution at least once. The do while loop in Java is a post-test loop in Java. Before checking the condition, it runs the loop's body. The process continues if the condition is satisfied, and the loop body is run once more. Until the condition is evaluated as false, the loop keeps running. When you want to guarantee that the loop body gets run at least once, regardless of the condition, the do-while loop is helpful. Let's look at the syntax for a do-while loop.

Syntax:	Flowchart:	Flowchart:
initialization; do { // Loop body; Statement(s); Increment/decrement; }while(test condition);		 Fig: do while loop flowchart

In the above syntax, the keyword "do" is followed by a set of curly braces that encloses a block of code or statements. After "do" block, there is a "while" keyword followed by a conditional expression in parentheses. This condition is a boolean expression that takes decision whether the loop should continue executing or not. If the condition evaluates to true, the loop will repeat to execute the code block. Otherwise, it will exit the loop. In situations where we always want to execute the code block at least once, we can employ the do while loop. Java's do while loop is a variant of the while loop that executes the code block once, before checking if the condition is true. It will then repeat the loop as long as the condition is true.

Explanation: From the do-while loop flowchart, it is clear that the loop body executes first, and then the loop conditional expression evaluates to determine whether to continue or terminate the loop. If the evaluation is true, the loop body executes again. This process continues until the test condition is true.

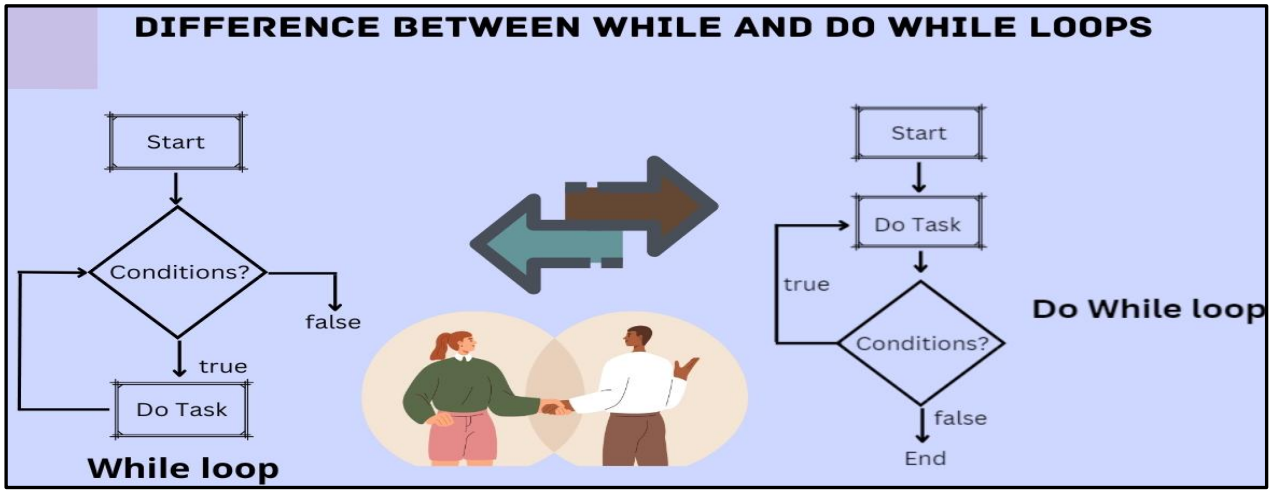
When the specified condition is false, the loop ends. The control of execution comes out of the do-while loop and goes to the next statement that appears immediately after the while statement.

Since the specified condition testes at the bottom of the loop, the do…while loop provides an exit-controlled loop. The test condition must be boolean expression.

Difference between While and Do While Loops:

While and do while loops are defined as control structures that repeat a block of code until the condition that is given is true. The main difference between the two loops is that the while loop checks the condition before the execution of the statement(s) whereas the do-while loop ensures that the statement(s) are executed at least once before evaluating the condition.

While Loop	Do-While Loop
In the While loop, the condition is tested before any statement is executed.	In Do while-loop, the statement is executed at least once even if the condition is false.
Syntax: while(condition){ // statements }	Syntax: do{ //statements }while(expression);
In While loop, no semicolon is needed after the end of the condition.	In Do-while loop, semicolon needed after the end of the condition
While loop is an entry-controlled loop.	Do-while loop is an exit-controlled loop.



Differences Between While, Do-While and For Loop:

For	While	Do-While
When loop has known number of iterations.	When number of iterations is unknown.	When loop body must be executed at least once.
It is known as exit-controlled loop.	It is known as entry-controlled loop.	It is known as entry-controlled loop.
Even if the condition is not true for the first time the control will enter in a loop.	If the condition is not true first time than control will never enter in a loop.	If the condition is not true first time than control will never enter in a loop.
Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax.
For loop is use when we know the number of iterations means where the loop will terminate.	While loop is use when we don't know the number of iterations means where the loop will terminate.	Do while loop is use when we don't know the number of iterations means where the loop will terminate.
Syntax: for (initialization; condition;updating); { Statements; }	Syntax: while(condition), {Statements; }	Syntax: do { Statements; } While(condition);
There is no semicolon; after the condition in the syntax of the for loop.	There is no semicolon; after the condition in the syntax of the while loop.	There is semicolon; after the condition in the syntax of the do while loop.
The control will never enter in a loop, if the condition is not true for the first time.	The control will never enter in a loop, if the condition is not true for the first time.	The control will enter a loop, even if the condition is not true for the first time.
The loop body does not execute if the condition is already false at the beginning (Entry controlled loop)	The loop body does not execute if the condition is already false at the beginning (Entry controlled loop)	The loop body executes once even if the condition is already false at the beginning (Exit controlled loop)

d) Nested For Loop in Java:

A nested loop has one loop inside of another. It allows you to repeat a set of instructions multiple times within another loop. When a loop is nested inside another loop, the inner loop is running many times inside the outer loop. In each iteration of the outer loop, the inner loop will be re-started. The inner loop must finish all of its iterations before the outer loop can continue to its next iteration. These are typically used for working with two dimensions such as printing stars in rows and columns.

Nested for loop in Java means one for statement inside another for statement. In other words, a for loop nested inside another for loop is called nested for loops. A nested for loops consists of an outer for loop and one or more inner for loops. Each time the outer for loop repeats, the inner for loop re-enters and starts a new execution. That is, each time the control will enter inside the inner for loop when the outer for loop repeats. We can put many loops inside a loop. But it is advice that you do not go beyond three levels of nested loops, as it will make the program look clumsy.

Syntax:

```
for (OuterInitialValue; OuterCondition; increment/decrement)  // Outer for loop.
{
for (InnerInitialValue; InnerCondition; increment/decrement)  // Inner for loop.
{
// statement of inner loop(body)
}
// statement of outer loop(body)
}
```

- The above syntax uses the for keyword with the small braces() containing all the values of the arguments.
- The initial value is the starting point for the loop, the condition is the endpoint, and increment/decrement is the steps taken to reach the condition.
- Firstly, the Outer loop will execute with the OuterInitialValue and then start the iterations of the inner loop until the InnerCondition becomes false.
- At that moment, the control will be transferred to the outer loop with the incremented or decremented value.

Flowchart:

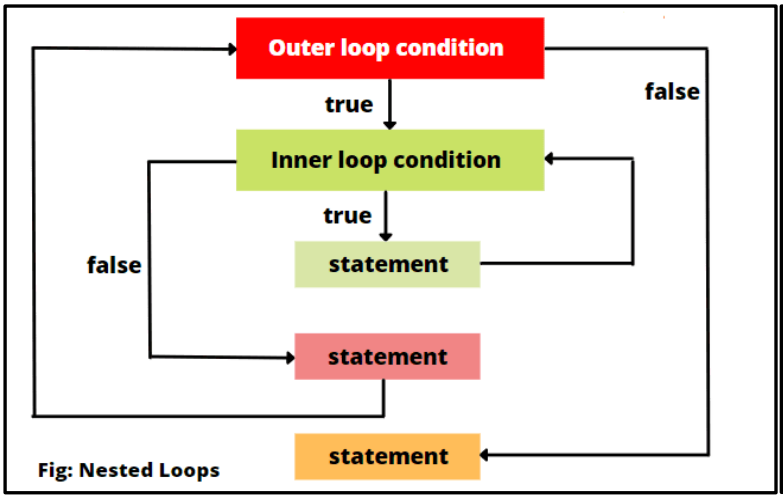
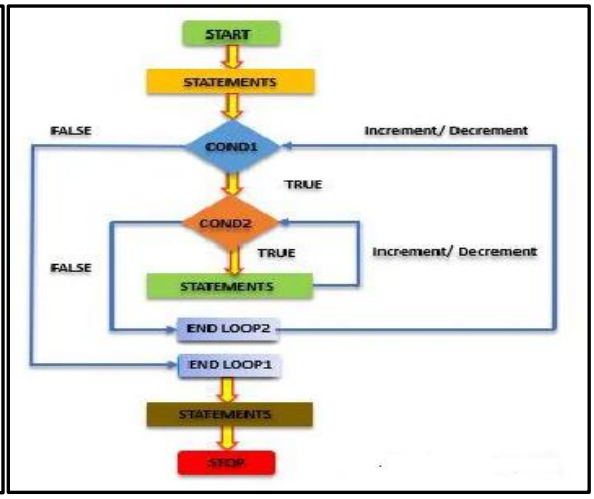


Fig: Nested Loops



Explanation: In the above flowchart, first, when we enter the body of the program, a statement such as initialization or print statements get executes. Once a loop is found, the program checks for the condition for the outer loop; if it returns true, it enters the loop; otherwise, the loop is ended, and the rest of the program's statements after loop executes.

Once it enters the outer loop and encounters the inner loop, variables are initialized if any is present, followed by checking the condition for inner loop if it returns true program enters into the inner loop; otherwise, go back to end of Loop1 and perform increment/decrement for executing Loop1 again.

In case cond2 returns true, Loop2 statements get executed, and the counter variable gets incremented/decremented. This procedure is repeated a number of times, and then the program exits from Loop2, then Loop1 and move to statements present after the loop.

Example Program:

```
for (int i = 1; i <= 3; i++)
{
statement1; // will execute 3 times.
for (int j = 1; j <= 4; j++)
{
statement2; // will execute 12 (3 * 4) times.
}
}
```

- In this example, when i = 1, execution will start from the outer for loop and statement1 will execute once.
- Now, the control of execution enters inside the inner for loop. Since we initially set the control variable j to 1, statement2 will execute once.
- After this execution, the value of j will be 2 because of increment by 1. Then, statement2 will execute once again.
- Like this, the inner for loop will execute 4 times with changing j values from 1 to 4. This means that statement2 will execute 4 times in the first execution of outer for loop.
- When the execution of inner for loop completes, the control of execution goes to the outer for loop. Now, the value of i will be 2 because of increment by 1.
- This time, the control of execution again enters inside the inner for loop and statement2 will execute 4 times.
- Then, the control again goes to outer for loop and the value of i will set to 3. Again, the inner for loop will execute 4 times.

This means that the values of i and j will change as:
When i = 1, j = 1, 2, 3, 4
i = 2, j = 1, 2, 3, 4
i = 3, j = 1, 2, 3, 4
In the above sequence, the outer for loop will execute total 3 times. Hence, statement1 inside the loop body will also execute 3 times.
Since the inner for loop will execute 4 times for each i value therefore, statements2 inside the loop body will execute 12 times.

For-each Loop in Java [Arrays]:

The for each loop in Java is also referred to as enhanced for loop. This feature is specially designed to retrieve elements of the array efficiently rather than using array indexes. Enhanced for loop repeatedly executes a group of statements for each element of the collection. It can execute as many times as a number of elements in the collection. In this for each loop we do not need provide initialization, condition, updating. This for each loop will use only in arrays and collections not in normal logical programs.
We can also print the Java array using for-each loop. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop. A "for each" loop in Java is a type of loop that added in Java version 5.0. It allows you to iterate through the elements of an array or collection without using an index variable. "For each" is translated from English as "for everyone". Actually, the foreach phrase itself is not used in this loop.

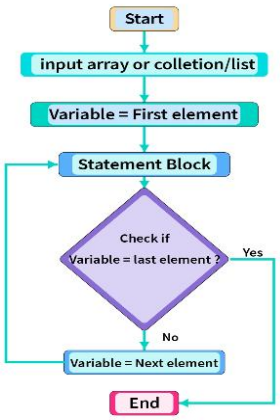
Syntax:

```
for(arraydatatype variablename : arrayname )
{
variablename//it holds each value of array at a time
}
```

- **arraydatatype** denotes the data type of the array.
- **variablename** denotes a variable assigned to each element in the array during the iteration.
- **arrayname** denotes the array in which the variable will iterate to execute the statements inside the loop block.

For each iteration, the for-each loop takes each element of the array and stores it in a loop variable. Thus, it executes the code written in the body of the loop for each element of the array or collection. Most importantly, the traversal happens until the last element of the array or collection. For-each loop in Java works like any other loop. The loop traverses for each element of the array till last. The value of the array element is stored in the variable declared with the loop, and execution of the statement occurs for each iteration.

Flowchart:



The working of the for-each loop is very interesting. We create a variable in the for-each loop's round brackets(), that stores each element of the iterable during each iteration.

The code written inside the for-each loop is executed for that particular element which is stored in the for-each loop variable and in the same way, the code is executed for all the elements of the iterable one by one. It means that the code that is written inside the body of for-each loop runs for all the iterable elements sequentially.

Example Program:

```
public class Main {
 public static void main(String args[]) {
String companies[] = {"Amazon", "Microsoft", "Apple", "Meta"};
for (String elem: companies)
{
  System.out.print(elem + " ");
} } }
```

Output

Amazon Microsoft Apple Meta

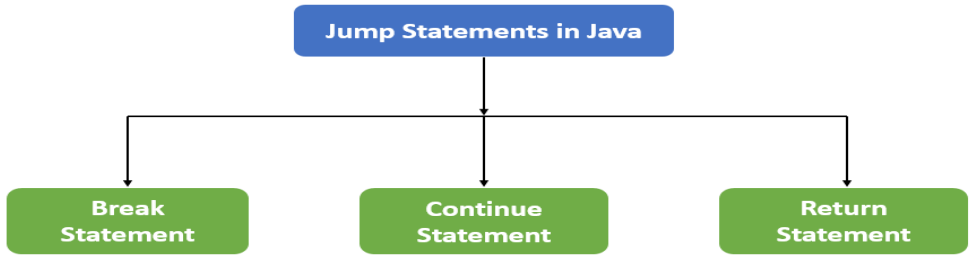
- In the above given example, we used for-each loop to iterate through all of the elements of the array called companies.
- In the variable elem, all the elements of the array companies get stored one by one which means that in each iteration the value of the iteration variable changes to the next element of the array.
- Then the code inside the for-each loop's body gets executed for-each of the element.
- Like in the 1st iteration elem = "Amazon", in the 2nd iteration elem = "Microsoft", in the 3rd iteration elem = "Apple" and similarly in the 4th iteration elem = "Meta".

3) Jumping Statements:

Jump statements in the Java programming language allow programmers to alter the normal flow of execution in their code. These statements enable efficient control flow by providing options to break out of loops, skip iterations, transfer control to specific points, and terminate functions. Jump statements control the program's flow by transferring execution to various code sections, which is essential for managing loops, switch cases, and functions. In Java jump statements are mainly used to transfer control to another part of our program depending on the conditions. Jumping statements are control statements that transfer execution control from one point to another point in the program. The jumping statements are the control statements which transfer the program execution control to a specific statement. Jump statements are primarily used to interrupt loop or switch-case instantly. When the flow of execution jumps to another part of code without carrying out any conditional test, it is called unconditional statements or unconditional execution in Java.

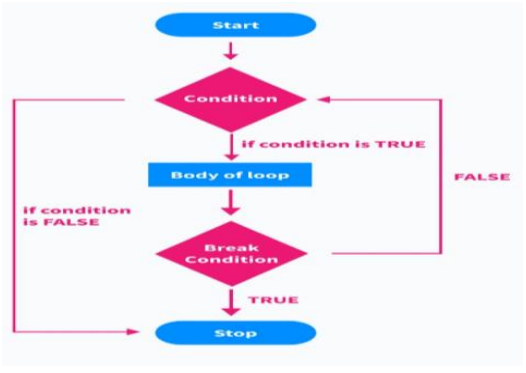
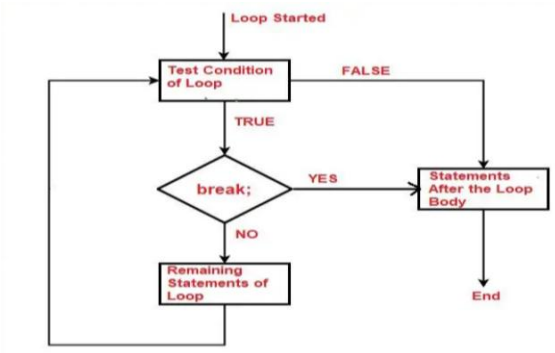
Jump statements are used to alter the normal flow of control within a program. They allow you to make decisions about which code to execute and when to exit loops prematurely.

Types of Jump Statements in Java:

1) Break Statement. **2)** Continue Statement. **3)** Return Statement.	

1) Break Statement:

In java, the break statement is used to exit from a loop prematurely. The break statement is useful when you want to exit from a loop or switch statement before it has finished executing all iterations. The break statement is used to terminate the execution of the nearest looping statement or switch statement. The break statement is widely used with the switch statement, for loop, while loop, do-while loop. When a break statement is found inside a loop, the loop is terminated, and the control reaches the statement that follows the loop.

Syntax:	**Flowchart:**	**Flowchart:**
if(conditition) { break; }		

The break statement in Java is primarily used to terminate the loop or switch statement in which it is present. As soon as the break statement is encountered within a loop, the loop is immediately terminated, and the program control resumes at the next statement following the loop.

You must follow some rules while using break statement:

- You must write break statement inside a loop or switch. Use of break keyword outside the switch or loop will result in compilation error.
- break statement is generally used with condition (if statements) inside loop.
- In case of nested loop or switch, break only terminates the innermost loop or switch.
- Use of break inside switch…case will transfer the program control outside the switch.
- break statement only terminates program control from nearest matched loop or switch.

Example Program1:

```
for(int i = 0; i < 10; i++) {
    if(i == 5) {
        break;
    }
    System.out.println(i);
}

# Output:
# 0
# 1
# 2
# 3
# 4
```

In this example, the break statement is used to exit the loop when i equals 5. The loop was supposed to print numbers from 0 to 9, but due to the break statement, it only prints numbers from 0 to 4.

The break statement can also be used in switch statements. In a switch statement, break is used to exit the switch case and resume the next line of code after the switch statement. If a break statement was not used in a switch case, all cases after the correct one would be executed until a break is encountered.

Example Program2:

```
int day = 3;
switch(day) {
     case 1:
          System.out.println("Monday");
          break;
     case 2:
          System.out.println("Tuesday");
          break;
     case 3:
          System.out.println("Wednesday");
          break;
     default:
          System.out.println("Invalid day");
}

# Output:
# Wednesday
```

In this example, the break statement is used to exit the switch statement after the correct case is executed. Without the break statement, if the day was 2, the output would be "Tuesday", "Wednesday", "Invalid day".

2) Continue Statement:

The continue keyword is generally used to end the current iteration in a (for, while, do...while etc).loop and jumps to the next iteration. It can be particularly useful when you want to skip a specific iteration rather than entirely breaking out of the loop. A loop control structure uses a Java continue statement to jump to the next iteration immediately. Within a loop in Java, once a continue statement is encountered, the control jumps directly to the start of the loop for the next iteration rather than executing the statements of the current iteration. Java continue statement is allowed only inside a loop body. When continue executes, the current iteration of the loop body terminates, and execution continues with the next iteration of the loop. In case of an inner loop, it continues the inner loop only. The continue statement pushes the next repetition of the loop to take place, hopping any code between itself and the conditional expression that controls the loop.

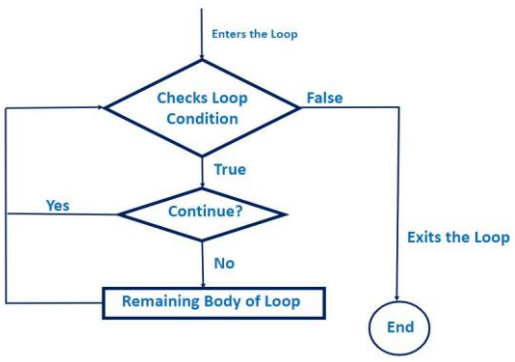
Example (Incoming call and an Alarm):

- Suppose we are on a call and the alarm is scheduled during the call time, then the alarm trigger recognizes the call event.
- Once the call event is noted, the phone continues the alarm to ring at the next snooze period. This is how continue essentially works.

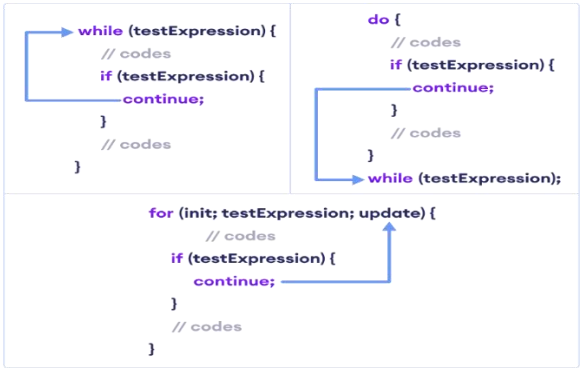
Syntax:

```
if(condititon)
{
continue;
}
```

Flowchart:



Flowchart:



when a continue statement is encountered the control directly jumps to the beginning of the loop for the next iteration instead of executing the statements of the current iteration. The continue statement is used

when we want to skip a particular condition and continue the rest execution. Java continue statement is used for all type of loops but it is generally used in for, while, and do-while loops.

Explanation: In this flowchart, the code will respond in the following steps:

- First of all, it will enter the loop where it checks the condition. If the loop condition is false, it directly exits the loop.
- After that it will check for 'continue' statement. If it is present, then the statements after that will not be executed in the same iteration of the loop.
- If 'continue' statement is not present, then all the statements after that will be executed.

Example Program:

```
for (int i=0; i<=5; i++){
if (i==3){
continue; //skips the rest of the statement
}
System.out.println(i);
}
Output:
0
1
2
4
5
```

The above output displays all the numbers except 3. When the continue statement is encountered, it jumps to the next iteration, skipping the statements for current iteration. It simply skips the rest of the code whenever "continue" construct is used.

Difference between break and continue:

Break	Continue
Exits from a loop or switch block immediately.	Skips the current iteration of a loop and proceeds to the next iteration.
Stops the loop, no further iterations occur.	Continues with the next iteration, the loop counter is incremented.
Exiting a loop when a specific condition is met.	Skipping iterations based on a specific condition (e.g., skipping even numbers in a loop).
Can make code easier to understand by preventing unnecessary iterations	Can improve readability by skipping unwanted iterations
May improve performance by terminating early	May improve performance by skipping unnecessary processing
break keyword is used to indicate break statements in java programming.	continue keyword is used to indicate continue statement in java programming.
We can use a break with the switch statement.	We cannot use a continue with the switch statement.
It stops the execution of the loop.	It does not stop the execution of the loop.

3) Java Return Statement:

In Java, return is a keyword that is used to exit from the method only with or without returning a value. Every method is declared with a return type in java and it is mandatory for Java methods. Return type may be a primitive data type like int, float, double, a reference type, or void type which represents "return nothing". That is, they don't give anything back. When a method is called, the method may return a value to the caller method. Return statement in Java is used to return a certain value from a code block when the execution n of that block is completed. The return statement exits a method and returns a value. The return statement is particularly useful when you want to provide a result or outcome from a method's operation. A method can accept data from outside and can also return the results. To return the result, a return statement is used inside a method to come out of it to the calling method.

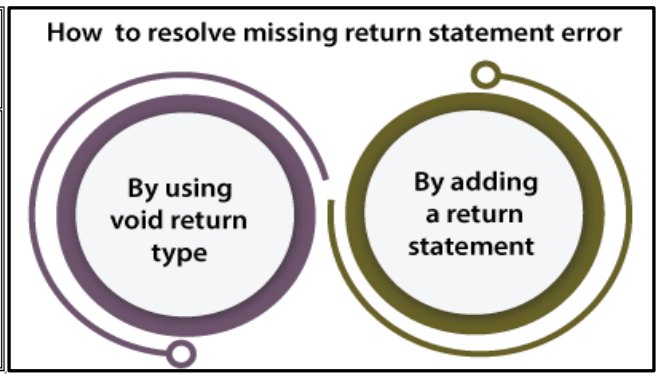
A return statement returns control of the current method or constructor to the caller. If a return statement does not contain an expression, the statement must be in a method declared with the void return type or in a constructor. Otherwise, the compiler issues an error message. If a return statement contains an expression,

it must be in a method that returns a value. The value produced by the expression is the return value of the current method.

Syntax:

```
return returnvalue;
```

```
public static String getHelloMessage()
{
      return "Hello World";
}
```



- In Java, every method is declared with a return type such as int, float, double, string, etc.
- These return types required a return statement at the end of the method. A return keyword is used for returning the resulted value.
- The void return type doesn't require any return statement. If we try to return a value from a void method, the compiler shows an error.
- The return type of the method and type of data returned at the end of the method should be of the same type. For example, if a method is declared with the float return type, the value returned should be of float type only.
- The variable that stores the returned value after the method is called should be a similar data type otherwise, the data might get lost.
- If a method is declared with parameters, the sequence of the parameter must be the same while declaration and method call.

****

Arrays in Java

Arrays in Java are non-primitive data types that store elements of a similar data type in the memory. Arrays in Java can store both primitive and non-primitive types of data in it. An array is a homogenous non-primitive data type used to save multiple elements (having the same data type) in a particular variable. An array refers to a data structure that contains homogeneous elements. Arrays are fixed in size and their elements are ordered. This means that all the elements in the array are of the same data type. According to the Java documentation, an array is an object containing a fixed number of values of the same type. The elements of an array are indexed, which means we can access them with numbers (called indexes). We can consider an array as a numbered list of cells, each cell being a variable holding a value. In Java, the numbering starts at 0.

[In this digital world, storing information is a very crucial task to do as a programmer. Every little piece of information needs to be saved in the memory location for future use. Instead of storing hundreds of values in different hundreds of variables, using an array, we can store all these values in just a single variable. To use that data frequently, we must assign a name, which is done by variable.**]**

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. There are primitive type arrays. This means we can use arrays of int, float, boolean, also arrays of String.

Array Creation in Java:

Syntax: Using the {} notation, by adding each element all at once:

datatype arrayvariable[]={value1,value2,valu3,…};	int[] age = {20, 21, 30};

Syntax: Using the **new** keyword, and assigning each position of the array individually.

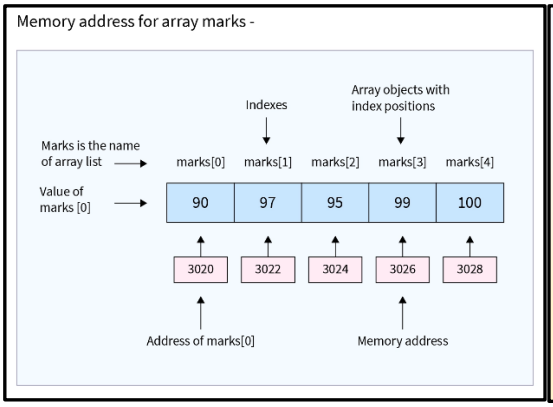
datatype arrayvariable[]= new datatype[size];		arrayvariable[index]=value;	
int[] marks = new int[3];	marks[0] = 50;	marks[1] = 70;	marks[2] = 93;

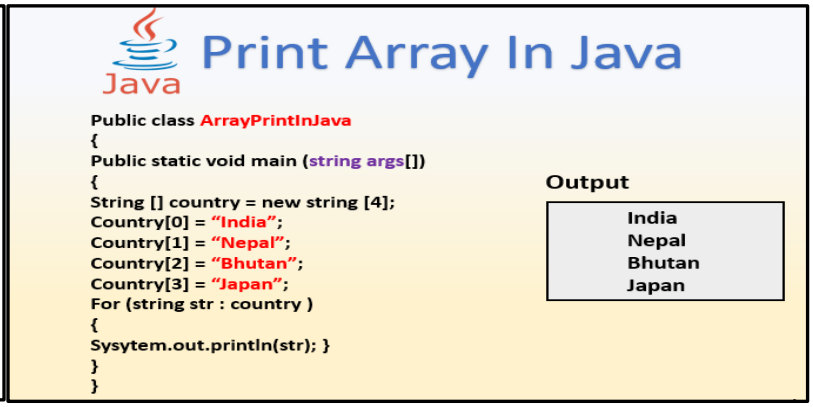
For Example:

You have to write a code that takes the marks of five students. Instead of creating five different variables, you can just create an array of lengths of five. As a result, it is so easy to store and access data from only one place, and also, it is memory efficient.

Without Array Declaration	With Array Declaration	Array Elements with Index
int mark1 = 90; int mark2 = 97; int mark3 = 95; int mark4 = 99; int mark5 = 100;	int marks [] ={78,84,63,91,75}	marks [0] = 90; marks [1] = 97; marks [2] = 95; marks [3] = 99; marks [4] = 100;

All above mentioned five variables could be accessed by the index given in the square bracket, which is known as index-based variables. The index always starts from 0 by default. So, if we have an array of 5 elements, it will have an index range from 0 to 4 by default. In the above example, the first element is stored at index 0 and the last one at the 4th index. The below-mentioned diagram shows how the array got placed in computer memory.





Advantages of Arrays in Java:

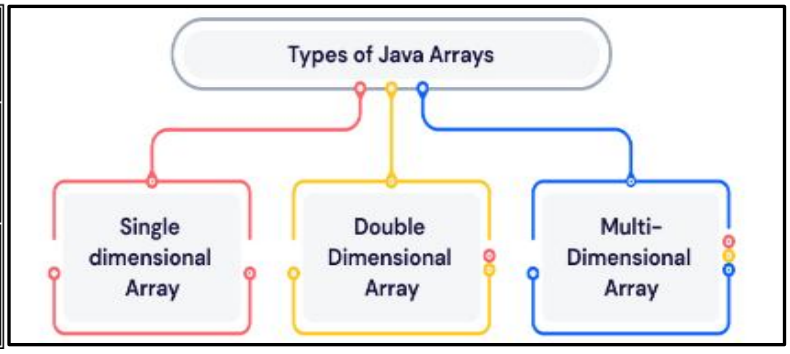
- Arrays are easy to declare, initialize, and use.
- Arrays are memory efficient as they can store multiple values in the same location. This reduces the amount of RAM required to store data and improves overall performance.
- Arrays support random access, meaning elements can be accessed directly using their index.
- The size of the array is fixed at the time of creation. This allows for better memory allocation and management.
- Arrays can be easily iterated using loops (for, while, enhanced for loop). This makes them useful for tasks that require repetitive operations.
- Arrays can also store variables and objects of class in Java.

Disadvantages of Arrays in Java:

- The size of the array cannot be increased or decreased once it is declared—arrays have a fixed size.
- Java cannot store heterogeneous data. It can only store a single type of primitives.
- If the array is initialized with a size larger than needed, it results in wasted memory.
- Arrays can only store elements of the same data type, which can be restrictive if different types of data need to be managed together.
- Arrays have a fixed size, which means the size must be known at compile-time and cannot be changed dynamically.

Types of Arrays in java:

1. One-Dimensional Array
2. Two-Dimensional Array
3. Multi-Dimensional Array



1. **One-Dimensional Array:**

A single dimensional array in Java is an array that holds a sequence of elements, all of the same type, accessed through an index. These elements are stored in contiguous memory locations, making it easy to access and manipulate them. A dimension is nothing but either, row a column. One dimensional array can be of either one row and multiple columns or multiple rows and one column. One dimensional array can store multiple values of the same primitive type such as int, float, long, String, etc. or objects of the same class. For instance, if we create an array to store integer data type values, all elements of the array must be integers, and they are accessed using their index positions. The index usually starts from zero and continues to the length of the array minus one. Each element in the array is assigned a unique index, starting from 0

for the first element, 1 for the second, and so on. This index is crucial for accessing and modifying array elements.

Declare and Initialize One-Dimensional Array:

There are basically **two** ways to create a one-dimensional array in java that are as follows:

1) The first way to declare single dimensional array with storing elements at the time of its declaration. We can declare one-dimensional array and store values (or elements) directly at the time of its declaration, like this:

```
int marks[ ] = { 90, 97, 95, 99, 100 }; // declare marks[ ] and initialize with five values.
```

In this single dimensional array declaration, **int** represents integer type values that can be stored into the array. **marks**[ ] defines the name of array with one dimension. A pair of square braces **[ ]** after array name represents one dimension. Then, we have mentioned marks obtained by five subjects (as elements) inside the curly **{ and }** braces. Now JVM will create five blocks represented by marks[0], marks[1], marks[2], and so on inside the memory with different addresses. Here, the values 0, 1, 2, 3, ... is called index value of array that refers to the element position in an array. In these blocks of memory, JVM stores five elements into an array, shown below:

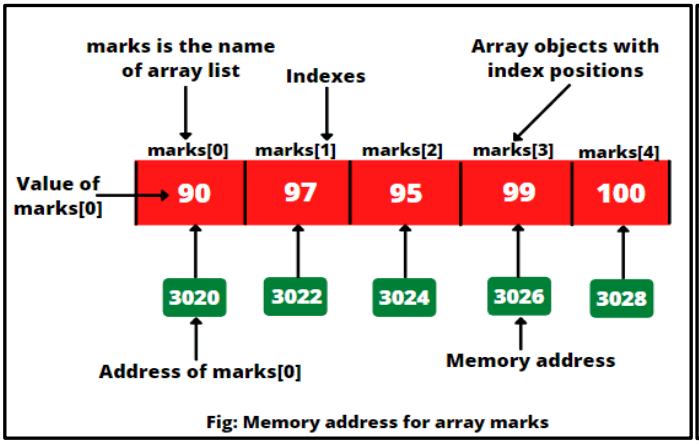
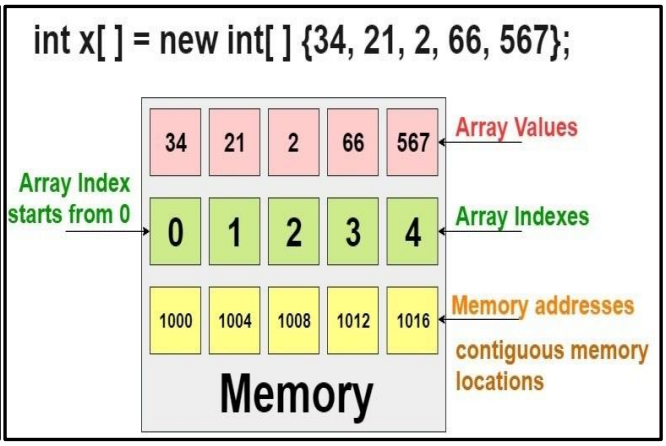


Fig: Memory address for array marks



2) The second way of creating a one-dimensional array is by declaring array first and then allocating memory for it using the new keyword.

```
int marks[ ] = new int[5]; // Allocating memory for storing 5 integer elements into an array.
```

JVM has allocated memory locations for storing five integer elements into the array. But we have not stored actual elements into the array so far. To store elements into the array, we can write statements like these in the program Or, the programmer can pass the values from the keyboard to the array by using for loop, as follows.

marks[0] = 90; marks[1] = 97; marks[2] = 95; marks[3] = 99; marks[4] = 100;	for(int i = 0; i < 5; i++) { marks[i] = scan.nextInt(); // Read the integer values from the keyboard and store them into marks[i]. }

Let us see some more examples for single dimensional array:
1. float salary[ ] = {99859.55f, 15000f, 14500.75f, 9050f}; 2. float salary[ ] = new float[10];
3. char ch [ ] = {'K','R','I','S','H','N','A'}; 4. char ch [ ] = new char [10] ;
5. String colors [ ] = {"Orange", "Red", "Sky blue", "Green"}; 6. String colors = new String[5];

2. Two-Dimensional Array:

A two-dimensional array in java represents many rows and columns of data. Data in matrix or table form can be represented by using two-dimensional array. A two-dimensional array is an array of arrays i.e., it's a collection of arrays in which elements are arranged in rows and columns (tabular format). We can access the elements using both the row index and column index. Java treats the two-dimensional array as a collection of multiple one-dimensional arrays. Hence, when we create a 2D array object, Java creates a simple one-dimensional array that references (points to) other one-dimensional arrays, which form the rows of the 2D array.

For example, marks obtained by a group of three students in five different subjects can be represented in table form like this:

Students	Physics	Chemistry	Maths	Computer	English
Deep	80	97	80	99	82
Amit	99	98	100	81	97
Larry	87	99	93	95	97

We can store marks of three students in the above table using two-dimensional array as:

int marks[ ] [ ] = { {80, 97, 80, 99, 82}, {99, 98, 100, 81, 97}, {87, 99, 93, 95, 97} };

The above marks obtained by three students in five subjects represent two-dimensional array (2D array) since it is a combination of 3 rows (no. of students) and 5 columns (no. of subjects). We can consider each row itself as a one-dimensional array. This two-dimensional array (2D array) contains three one-dimensional arrays.

Declare and Initialize Two-Dimensional Array:

There are mainly **two** ways to create a two-dimensional array in java that are as follows:

1) To create a two-dimensional array in Java, you have to specify the data type of items to be stored in the array, followed by two square brackets and the name of the array. The syntax to create a two-dimensional array and directly store elements at the time of its declaration, **as follows**:

int marks[ ] [ ] = {{50, 60, 78, 87, 95}, **{98, 87, 75, 76, 99},** **{98, 99, 97, 95, 100}};**	Here, **int** represents the types of elements stored in an array. We have stored integer type elements in this array. **marks** represent array name. Two pairs of square braces (**[ ] [ ]**) after array name represents a two dimensional array. Elements of each row must be written inside the curly braces { and }.The rows and elements in each row must be separated by commas (,).

Since there are three rows and five columns in each row, JVM creates 3 * 5 = 15 memory blocks, as there are 15 elements being stored into the array. These blocks of memory can be individually referred to as:

marks[0][0], marks[0][1], marks[0][2], marks[0][3], marks[0][4]
marks[1][0], marks[1][1], marks[1][2], marks[1][3], marks[1][4]
marks[2][0], marks[2][1], marks[2][2], marks[2][3], marks[2][4]

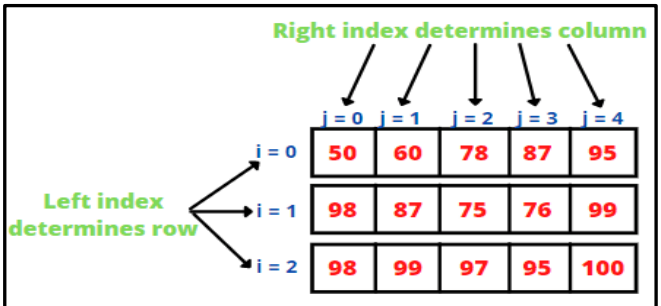


Fig: Arrangement of elements in two dimensional array

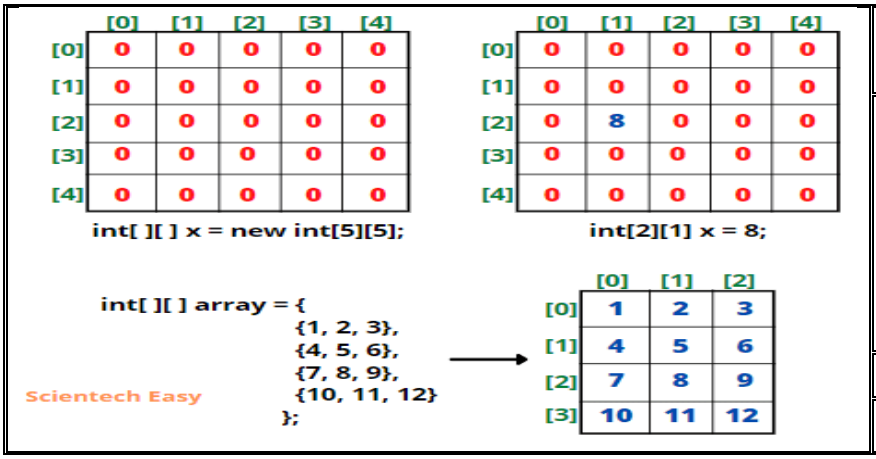
As you can observe that rows are starting from 0 to 2 and columns are starting from 0 to 4. So, in general form, any element can be referred to as marks[i][j] where i and j are two indexes that represent row position and column position respectively. In a two-dimensional array, we access an element through a row and column index.

2) The second way for creating a two-dimensional array is that first declare an array and then allocate memory for it using the new operator.

The syntax to declare a two-dimensional array 3-by-5 of type int is as follows:

```
int[ ][ ] marks = new int[3][5];
```

Here, JVM allocates memory for storing 15 integer values into the array. But so far, we have not stored 15 values into an array. We can store elements into a two-dimensional array by accepting them from the keyboard and from within the program also.



Let us see some more examples for double dimensional array:

```
float[ ] [ ] x = {{1.1f, 1.2f, 1.3f },
                  {2.1f, 2.2f, 2.3f},
                  {3.1f, 3.2f, 3.3f}
                 };
```

(or)

```
float[ ][ ] values = new float[3][4];
```

```
String[ ][ ] str = new String[ 3][3];
```

```
int[ ][ ] values = new int[3][4];
```

****

Strings in Java

Strings are used for storing text. A string is sequence of characters or collection of characters, here, characters mean combination of A-Z, a-z,0-9 and special symbols (@#$%^) including spaces. The string is a non-primitive data type. Declaring a string is as simple as declaring a one-dimensional array. We use double quotes to represent a string in Java. For example, "hello" is a string containing a sequence of characters 'h' , 'e' , 'l' , 'l' , and 'o'. Strings in Java are objects that can hold a sequence of characters contained within a pair of double quotes ("). In java the, java. lang. String class is used to create a Java string object. Below is the basic syntax for declaring a string.

2 Ways to create a string in Java:

1) Using String Literal:

This is the most common way of creating string. In this case a string literal is enclosed with double quotes. You can insert any text or character between double quotes. All of the content/characters can be added in between the double quotes.

Syntax: datatype variablename = "value";	**Ex:** String courseName = "Core Java";

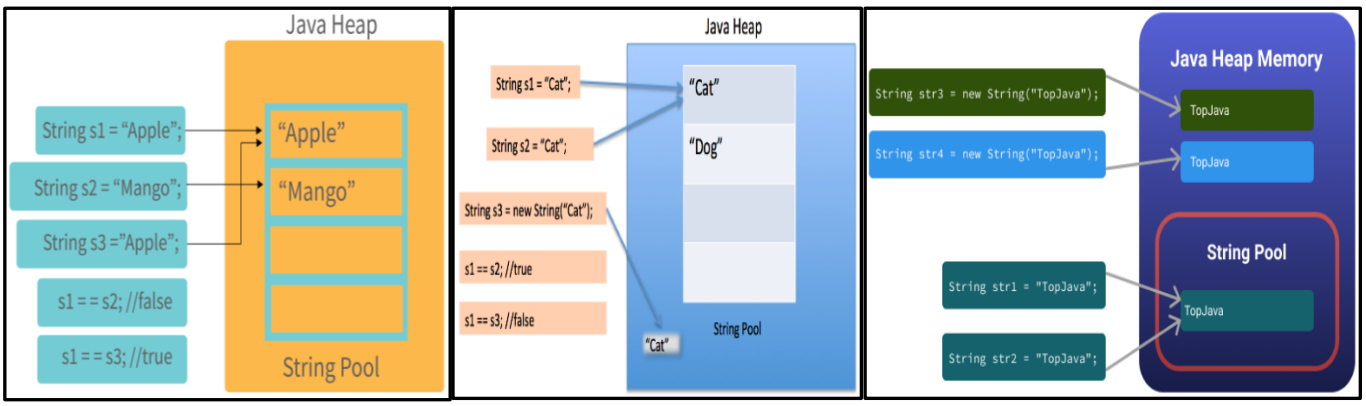
In the above-given example, we have declared a string variable called **courseName** and initialized it with the string value "**Core Java**".

Now we will see the memory management of Strings in Java. The Strings in Java are stored in a special place in heap called "String Constant Pool" or "String Pool".

String Pool in Java:

Memory management is the most important aspect of any programming language. Memory management in case of string in Java is a little bit different than any other class. To make Java more memory efficient, JVM introduced a special memory area for the string called String Constant Pool. When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.

String s1="Core Java";	**String s2="Core Java";**
When you are declaring a string literal(i.e without using the new keyword), the JVM stores this string in a pool of strings called the String Constant Pool. Simple,right? Now we declared two strings of the same value "Core Java" in the case above. So what JVM does is, it creates a reference **s1** and points it to the object "Core Java". Whenever the compiler executes the second line which is String **s2**=" Core Java"; it realizes that Core Java object is already in the String Constant Pool ! So, it points the new instance s2 to the same old "Core Java" object created in the first line. No new objects create. This results in saving up space as the JVM did not need to create a different object for the same String. That is why the String Constant Pool is an integral part of Java.	



In the above image, two Strings are created using literal i.e "Apple" and "Mango". Now, when third	While creating the s1, JVM will not find any string object having the value "Cat" in the string constant

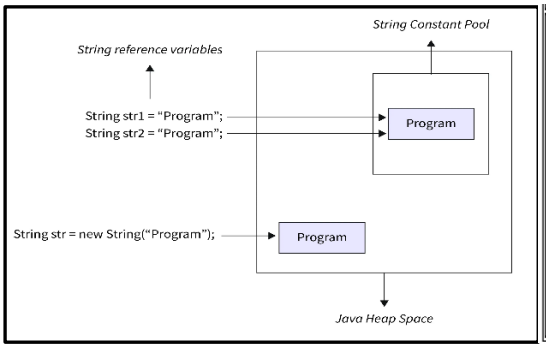
String is created with the value “Apple”, instead of creating a new object, the already present object reference is returned. That’s the reason Java String pool came into the picture.	pool to create a new object. For s2, it will find the string with the same value in the pool. Thus, no new object will be created, and the reference to the previous instance will be returned.

2) Using new keyword or String Objects:

Java String is created by using a keyword “new”. The new keyword always creates a new String object in the heap memory, regardless of whether an identical string already exists in the string pool. Strings can indeed be created using the new keyword in Java. When a string is created with new, a new object of the String class is created in the heap memory, outside the string constant pool. The new keyword should be used when explicitly needed to create distinct string objects.

Syntax: datatype objectname =new datatype (“Value”); **Ex:** String str = new String(“Core Java Class”);
String newStringOne = new String("Java"); **String newStringTwo = new String("Java");** Although the value of both Strings will be the same as earlier, we’ll have to different objects this time.

Unlike string literals, these objects are allocated separate memory space in the heap, regardless of whether the same value already exists in the heap or not.



In this code snippet, how many String objects are created?

String str1 = "Program";

String str2 = "Program";

String str = new String("Program");

Answer: Two String objects are created. str1 and str2 both refer to the same object and str has the same content but using new keyword, forced the creation of a new, distinct, object.

“Main difference between String literal and String object is that, in case of String literal new instance will not be created if string value is already present in the string constant pool but in case of string object, a new string instance will be created whether the value is present in the string pool or not.”

Heap Memory	Stack Memory
The heap is a region of memory used for dynamic memory allocation. It is where objects are created and reside during the lifetime of a Java program. The heap is shared among all threads, and memory is allocated and deallocated by the Java Virtual Machine (JVM) automatically through a process known as garbage collection.	The stack is a region of memory used for method calls and local variable storage. Each thread in a Java program has its own stack, which is created along with the thread. The stack operates in a Last-In-First-Out (LIFO) manner, where the most recently added item is the first to be removed.
This memory, in contrast to stack, isn’t automatically deallocated. It needs Garbage Collector to free up unused objects so as to keep the efficiency of the memory usage.	When the method finishes execution, its corresponding stack frame is flushed, the flow goes back to the calling method, and space becomes available for the next method.
Heap memory is used by all the parts of the application	Stack memory is used only by one thread of execution.
Heap Memory: Objects remain in memory until they are no longer referenced and the Garbage	Variables are destroyed once the function call is complete.

Collection removes them.	
Managed dynamically, with variable sizes depending on object size.	Managed by the thread, with a fixed allocation size.
String str = new String("Hello"); - the string object is created on the heap, but the reference is stored on the stack.	String str = "Hello"; The reference to the string literal is stored on the stack.
Objects stored in the heap are globally accessible.	Stack memory can't be accessed by other threads.

Java String Methods:

1. **String length() Method:**

The length() method returns the length of a specified string.
Syntax: public int length();
Returns: Length of characters. In other words, the total number of characters present in the string.
Ex: String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("Given String length is"+txt.length());
Output: Given String length is 26.

2. **String copyValueOf() Method:**

The method copyValueOf() is used for copying an array of characters to the String. The point to note here is that this method does not append the content in String, instead it replaces the existing string value with the sequence of characters of array.
static copyValueOf(char[] data): It copies the whole array (data) to the string.
Syntax: public static String copyValueOf(char[] data)
Parameters: data − the character array.
Return Value: This method returns a String that contains the characters of the character array.
char[] Str1 = {'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'};
String Str2 = " ";
Str2 = Str2.copyValueOf(Str1);
System.out.println("Returned String: " + Str2);
Output: Returned String: hello world

3. **Java String toLowerCase() Method:**

The java string toLowerCase() method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.
Syntax: public String toLowerCase()
Return Value: The String, converted to lowercase.
Example: String s1="STRING LOWER CASE METHOD";
String s1lower=s1.toLowerCase();
System.out.println(s1lower);
Output: string lower case method

4. **Java String toUpperCase() Method:**

The java string toUpperCase() method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.
Syntax: public String toUpperCase()
Return Value: String in uppercase letter.
Example: String s1="upper case string";
String s1upper=s1.toUpperCase();
System.out.println(s1upper);
Output: UPPER CASE STRING

5. Java String replace() Method:

The Java String class replace() method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Syntax: public String replace(char oldChar, char newChar)

public String replace(CharSequence target, CharSequence replacement)

Parameters: oldChar : old character

newChar : new character

target : target sequence of characters

replacement : replacement sequence of characters

Returns: replaced string

Exception Throws: NullPointerException: if the replacement or target is equal to null.

Example1:

```
String s1="I am good in Java";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
```

Output: I em good in Jeva

Example2:

```
String s1="My trainer name was Krishna";
String replaceString=s1.replace("was","is");//replaces all occurrences of "was" to "is"
System.out.println(replaceString);
```

Output: My trainer name is Krishna

6. Java String trim() Method:

The trim() method removes whitespace from both ends of a string.

Note: This method does not change the original string.

Syntax: public String trim()

Returns: A String value, which is a copy of the string, without leading and trailing whitespace

Example:

```
String s8=" Hello World! ";
System.out.println(s8);//without trim()
System.out.println(s8.trim());//with trim()
```

Output:

Hello World!

Hello World!

7. **String concat() Method:**

The concat() method appends (concatenate) a string to the end of another string.

Syntax: public String concat(String string2)

Parameters:

string2: A String, representing the string that should be appended to the other string

Returns: A String, representing the text of the combined strings

Example:

```
String firstName = "Full Stack ";
String lastName = "Java";
System.out.println(firstName.concat(lastName));
```

Output: Full Stack Java

8. String isEmpty() Method:

The Java String class isEmpty() method checks if the input string is empty or not. Note that here empty means the number of characters contained in a string is zero. This method returns true if the given string is empty, else it returns false.

Syntax: public boolean isEmpty()

Returns: true if length is 0 otherwise false.
Example: String myStr1 = "Hello";
String myStr2 = "";
System.out.println(myStr1.isEmpty());
System.out.println(myStr2.isEmpty());
Output:
false
true

9. String equals() and equalsIgnoreCase() Methods:

- **equals() Method:** The equals() method is used to compare the content (value) of two strings.

It returns true if both strings contain exactly the same characters in the same order, including case sensitivity.

Syntax: string.equals(String str)

- **Parameters**
  - string → an object of the String class
  - str → the string to be compared
- **Return Values**
  - Returns true → if both strings are equal
  - Returns false → if strings are not equal
  - Returns false → if the argument str is null
- **Example: equals()**

String Str1 = "Hello";
String Str2 = "Hello";
String Str3 = "Hi";
System.out.println(Str1.equals(Str2)); // true
System.out.println(Str1.equals(Str3)); // false

Output:
true
false

- **equalsIgnoreCase() Method:** The equalsIgnoreCase() method compares two strings ignoring case differences.

Syntax: string.equalsIgnoreCase(String str)

- **Return Values**
  - Returns true → if strings are equal ignoring case
  - Returns false → if strings are not equal
  - Returns false → if the argument str is null
- **Example**: equalsIgnoreCase()

String s1 = "JAVA";
String s2 = "java";
System.out.println(s1.equalsIgnoreCase(s2)); // true

Output: true

10. String contains(): The contains() method checks whether a string contains a specific sequence of characters. It returns a boolean value.

✓ Case-sensitive

✓ Does not modify the original string (String is immutable)

Syntax: string.contains(CharSequence seq)

- **Parameter**

- seq → a CharSequence to search for

🔹 **Return Values**

- true → if the sequence is found
- false → if the sequence is not found
- ❌ NullPointerException → if seq is null

Example:

```
String str = "Java Programming";
System.out.println(str.contains("Java"));  // true
System.out.println(str.contains("Prog"));  // true
System.out.println(str.contains("java"));  // false
```

Output:

```
true
true
false
```

****

OOPS stands for "Object-Oriented Programming System" or "Object-Oriented Programming Structure." It is a programming paradigm based on the concept of "objects," which can contain data and code to manipulate that data. Object-Oriented Programming (OOPS) concepts in Java helps reduce code complexity and enables the reusability of code. Object-oriented programming is designed to organise code in a way that models real-world entities and their interactions, making it easier to understand, maintain software systems. The basic concept of OOPS is to create objects, which can hold both data and functions. Objects are instances of a class, a blueprint for an object. OOP promotes the concept of modularity by breaking down complex systems into manageable components called objects. In Java, objects encapsulate data and behavior, allowing for easier maintenance, code reuse, and efficient development.

Classes: A class is a blueprint for creating objects. It defines the properties (data members) and behaviours (member functions or methods) that the objects derived from the class will possess.

Objects: Objects are instances of classes, encapsulating data and behaviour. They represent real-world entities and provide a modular and organised way to design software.

Method: A method in Java is a block of code that, when called, performs specific actions mentioned in it. You can insert values or parameters into methods, and they will only be executed when called. A method in Java is a set of instructions that can be called for execution using the method name. A Java method can take in data or parameters and return a value - both parameters and return values are optional. Methods can be public, private or protected. **Types**: Predefined Method, User-defined Method [With returntype with parameter, Without returntype without parameter, With returntype without parameter, Without returntype with parameter].

Constructor: Constructor in java is used to create the instance of the class. Constructors are almost similar to methods except for two things - its name is the same as the class name and it has no return type. Sometimes constructors are also referred to as special methods to initialize an object. **Types**: Parameterized Constructor, No-Argument Constructor, Default Constructor.

Inheritance: Inheritance allows a class (subclass or derived class) to inherit properties and behaviours from another class (base class or parent class). This promotes code reusability and establishes an "is-a" relationship between classes. **Types**: Single Inheritance, Multilevel Inheritance, Hierarchical Inheritance, Multiple Inheritance, Hybrid Inheritance.

Method Overloading: Overloading a method, means creating a different method with the same name in the same class, but with a different parameter list. A method can also be overloaded by changing the number of parameters. Basically, we can create multiple methods with the same name in the same class. However, each method works differently. It makes the program easier to read (readable). Used to achieve **Compile time polymorphism. Types**: Changing the Number of Parameters, Changing Data Types of the Arguments, Changing the Order of the Parameters of Methods.

Method Overriding: Method overriding occurs when a subclass provides a particular implementation of a method declared by one of its parent classes. Method signature (name and parameters) is the same in the superclass and the child class, it's called overriding. The main purpose of having overridden methods in the child is having a different implementation for a method that is present in the parent class. Overriding method: The method in child class that is the same as a method in the parent class. Used to achieve **Runtime polymorphism.**

Polymorphism: Polymorphism is derived from two Greek words, "poly" and "morph", which mean "many" and "forms", respectively. polymorphism means many forms. Polymorphism allows us to perform a single action in different ways. Polymorphism allows you to define one interface and have multiple implementations. **Types**: Compile-time Polymorphism, Runtime Polymorphism.

Data Abstraction: Data abstraction involves exposing only essential features of an object while hiding the

implementation details. It allows programmers to focus on the functionality of objects without worrying about their internal workings.

Encapsulation: Encapsulation involves bundling data and methods that operate on the data within a single unit, i.e., a class. This provides data hiding and prevents direct access to the internal representation of an object.

****

Methods in Java

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation means which take input from the user, processed it and give output. It is used to achieve the reusability of code. A method is a block of code which only runs when it is called. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a block of code. The method is executed only when we call or invoke it. If any error occurs easily i can modify piece of code without disturbing other parts of code. Methods are also called as functions. It reduces the size of the program / project.

A Java method can perform some specific tasks without returning anything. Methods in Java allow us to reuse the code without retyping the code. A method is like function i.e. used to expose behavior of an object.

Method Declaration:

The method declaration provides information about method attributes, such as visibility, return-type, name, arguments and a pair of parentheses, (), and a body between braces, {}. In general, method declarations have six components:

Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

Access Specifier (Modifiers): Access specifier or modifier is the access type of the method. It specifies the visibility of the method.

Java provides Four types of Success Specifier:

Public: The method is accessible by all classes when we use public specifier in our application.

Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.

Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.

Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction (). A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma (,) and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It including the declaration of local variables. The code you need to be executed to perform your intended operations. It is enclosed within the pair of curly braces ({}).

Syntax: Declare a method

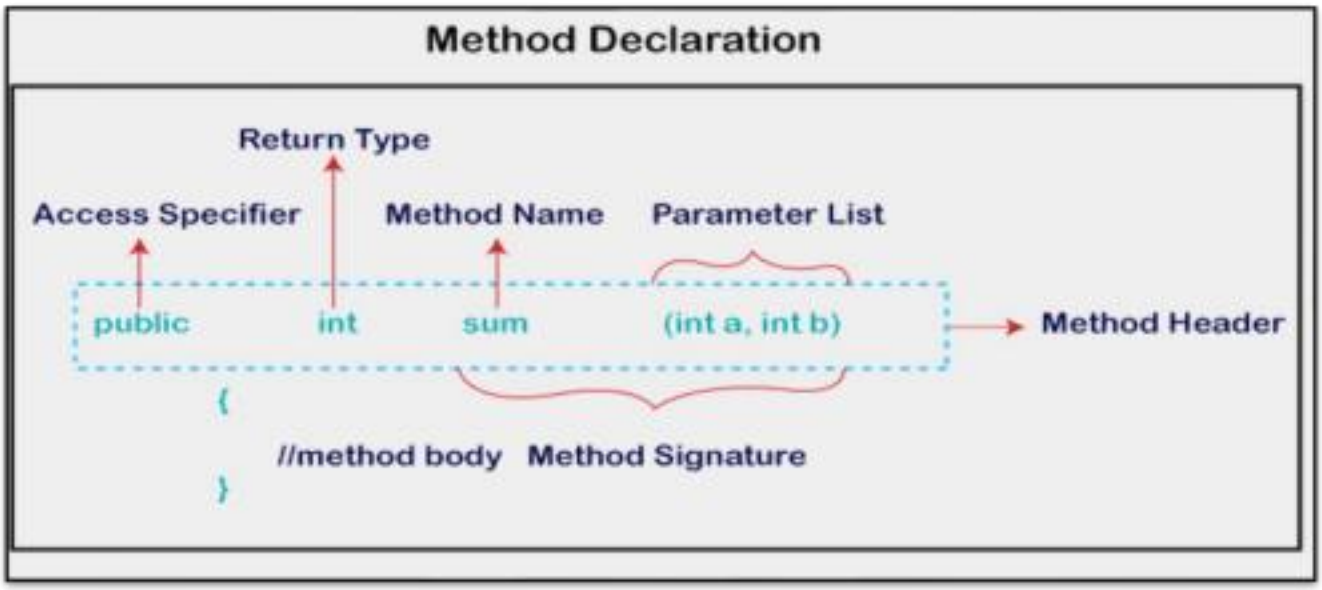
<access_modifier> <return_type> <method_name> (list_of_parameters) {
//method body
}

Example:

```
public int sum (int a, int b)
{
//method body
}
```

In the above **Example**:

'**public'** is the access specifier, return type is '**int'** (i.e. integer), method name is '**sum, int a, int b** are the parameters and sum (int a, int b) are the method signature.



Without Methods Concept	
Logics will be repeated	Modification will disturb whole code
Length of program is increased	No reusability (we cannot use code outside class)
Increased Code Complexity	Difficulty in Implementing Object-Oriented Principles

Naming a Method:

Method name must be a verb and start with a lowercase letter.

[Ex: sum(), divide(), area()]

If the method name has more than two words, the first name must be a verb followed by adjective or noun without any space.

In the multi-word method name, the first letter of each word must be in uppercase except the first word.

[Ex: addIntegers(), areaOfSquare, stringComparision()]

Try to use method name that corresponds to the functionality (if the method is adding two numbers, use add() or sum())

Example:

- int totalMarks ---> variable
- int totalMarks[] ---> array
- class totalMarks ---> class
- int totalMarks() ---> method

Note: It is also possible that a method has the same name as another method name in the same class, it is known as method overloading.

We can declare methods in two types:

1.Static Method **2.Non-Static Method**

1. Static Method:

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword static before the method name. The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the main () method.

2. Non-Static Method:(or) **Instance Method:**

Instance method are methods which require an object of its class to be created before it can be called. To invoke an instance method, we have to create an Object of the class in which the method is defined. A non-static method can access any static variable without creating an instance of the class because the static variable belongs to the class. In object-oriented programming (OOP), an instance is a specific realization of any object. The method of the class is known as an instance method. It is a non-static method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class.

Difference between Static Method and Non-Static Method:

Static Method	Non-Static Method
A static method belongs to the class	A non-static method belongs to an object of a class.
Static methods are useful if you have only one instance. [Ex: College]	Non-static methods are used if you're going to use your method to create multiple copies. [Ex: Students].
To create a static method in Java, you prefix the key word 'static' before the name of the method.	A non-static method in Java does not have the key word 'static' before the name of the method.
Referring to the dress analogy, a static method would get its design from the original dress.	From the dress example, a non-static method would get the measurements from a pattern of the original dress.
A static method can only access static variables; it cannot access instance variables.	A non-static method can access any static variable without creating an instance of the class because the static variable belongs to the class.
To access a non-static method from a static method, create an instance of the class	A non-static method can access any static method without creating an instance of the class
The static method cannot override in Java.	You can override a non-static or instance method

What is Static Block in Java?

In a Java class, a static block is a set of instructions that is run only once when a class is loaded into memory. A static block is also called a static initialization block. This is because it is an option for initializing or setting up the class at run-time.

Abstract Method: The method that does not has method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the abstract class. It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword abstract.

Syntax: abstract void method_name() ;

Components Of Functions:

Any function whether it is a system or user- defined function should contain three main elements. They are

- Function declaration or function prototype
- Calling function or function call
- Called function or function definition

Function Declaration: Function declaration indicates that the type of function to be developed by the user it means whether the function have arguments or not and function Return value or not.

Syntax: ReturnType function name [(Arguments-list)]

Ex: void fact(n);

Void sum ();

Calling Function: A function can be called by the name of the function and such statement is known calling function. Whenever you develop a function, to execute that function body that function must be calling.

Syntax: Function – name ([argument – list]);

Ex: fact(n); or Sum();

Called Function: The statement required for developing a user – define function must be specified in the called function or function definition. The statement written in the called function is called body of the function or function body.

Syntax:

```
Return-type functionname([argument-list]);
{
Statements;
}
Ex:
Void fact();
{
int m;
}
```

How to call a Method/ Method Calling?

In one Program if we write multiple methods and if we do not call those methods will not execute.

To call a method in Java, you have to write the method's name followed by parentheses () and a semicolon; Method we have to call or invoke explicitly.

For using a method in a program, it should be called. There are two ways in which a method is called i.e., the method returns a value or it returns nothing.

The process of method calling is simple. When a program invokes a method, the program control automatically gets transferred to the called method. This called method returns control to the caller in these two conditions, they are namely:

- When the return statement is executed.
- When the control reaches the method ending closing brace.

How to call methods Outside class?

A. Check access Modifier or Specifier:

Access modifier will tell us that is method visible or not outside class, in package or outside package.

Access modifier	Within class Within Package	OutsideClass with in Package	OutsideClass Outside Package
public	yes	yes	yes
private	yes	no	no
protected	yes	yes	no
default	yes	yes	no

B. Check Method is Static / Non-Static:

If method is static call with class name inside package
If method is static call with class name and import package.class , outside package
If method is non static call with Object name inside package
If method is non static call with Object name and import package.class , outside package

Types of Method:

There are two types of methods in Java:

1.Predefined Method 2. User-defined Method

1. Predefined Method: [Built-in Methods / Standard Library Methods / System defined Methods]**:**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method or System defined Methods. This means that they can be called and used anywhere in our program without defining them. We can directly use these methods just by calling them in the program at any point.

Some examples of Standard libraries (pre-defined methods) are **length (), equals (), compareTo(), sqrt(), min(),max() and print(), etc** and each of them is defined inside their respective classes. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

These standard libraries come along with the Java Class Library that is present in a Java archive (*.jar) file with JVM and JRE. The file stdlib.jar holds together all standard libraries into one single file.

Each and every predefined method is defined inside a class. Such as print() method is defined in the java.io.PrintStream class. It prints the statement that we write inside the method. For example, print("Java"), it prints Java on the console.

When we required any pre-defined method, we just call the method by its name.

2. User-Defined Method:

If we cannot achieve business requirements with predefined methods then we have to design our own methods that is known user define methods. The method written by the user or programmer is known as a user-defined method. These methods are created by the programmers for their requirement and modified according to the requirement.

Here's an example of a user-defined method.

Calling User-Defined Method in Java:

To call a user-defined method, first, we create a method and then call it. A method must be created in the class with the name of the method, followed by parentheses (). The method definition consists of a method header and method body.

Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method.

When we are writing method in class point to be remember?

- Do not write method definition inside main method or any other method.
- We can write methods inside class without main method but methods execution will not happen because java execution will start with main only. If we want to execute methods call inside main method.
- If the return type is void, the method can perform a task without returning any value. Such a method is called void method. There will be no return statement inside the method body.
- If the return type is non-void but a data-type like int, float, double, etc, the method will return a value.
- It will not be possible for execution to reach the end of the method body without executing a return statement.

Return Statement in Java:

In Java programming, the return statement is used for returning a value when the execution of the block is completed. The return statement inside a loop will cause the loop to break and further statements will be ignored by the compiler.

A return statement causes the program control to transfer back to the caller of a method. Every method in Java is declared with a return type and it is mandatory for all java methods. A return type may be a primitive type like int, float, double, a reference type or void type (returns nothing).

There are a few important things to understand about returning the values:

- The type of data returned by a method must be compatible with the return type specified by the method. For instance, if the return type of some method is boolean, we cannot return an integer.
- The variable receiving the value returned by a method must also be compatible with the return type specified for the method.
- The parameters can be passed in a sequence and they must be accepted by the method in the same sequence.

Syntax: return returnvalue;

Ex: return x;

Scope of Method:

In java we have four access modifier which decides scope of method, means where we can access method where we cannot.

public: In method signature if we found public modifier we can access method in class, in package, in project anywhere.

private: In method signature if we found private modifier, we can access methods only in class. outside classes cannot access.

protected: Package level (we can access in class, outside class in package).

default: Same as protected.

Note: No need to write default, if we do not write public private protected it is considered as default.

Parameters:

The parameters are the input values which will be passed to the function. It tells about the data types of the arguments, their order and the number of arguments that will be passed to the function. The parameters are optional. You can also have functions without parameters.

Parameters are Two Types:

1.Actual Parameter: Those parameters which are passed to functions while calling them is are known as actual parameter.

2.Formal Parameter: Those parameters which are received by the functions are known as formal parameters.

Types of User-Defined Methods in Java:

There are various kinds of user-defined functions based of the return type & arguments passed.

In Four ways we can write user define methods:

1) Without Returntype Without Parameter
2) With Returntype Without Parameter
3) Without Returntype With Parameter
4) With Returntype With Parameter

1) Without Returntype - Without Parameter: [Function with No argument and No return value**]**

A method without a return type and without parameters is typically used when you want the method to perform an action but do not need any input from the caller, nor do you need to return any value. The method is declared with the void keyword to indicate that it does not return anything. Function with no argument means the called function does not receive any data from calling function and Function with no return value means calling function does not receive any data from the called function. So, there is no data transfer between calling and the called function.

Syntax:

```
void Addition ()
 {
 Statements;
Body;
 }
```

2) With Returntype - Without Parameter: [Function with no arguments but returns a value**]**

A method with a return type and without parameters returns a value but does not accept any input arguments. Function with no arguments and one return value as said earlier function with no arguments means called function does not receive any data from calling function and function with one
return value means one result will be sent back to the caller from the function.

Syntax:

```
int Addition()
{
Statements;
Body;
return x;
 }
```

3) Without Returntype - With Parameter: [Function with arguments but no return value**]**

A method without a return type but with parameters is defined using the void keyword for the return type and specifies the parameters in the method signature. Such methods perform operations or actions but do not return any value. They can take one or more parameters as input to perform their tasks. When a function has arguments, it receives any data from the calling function but it returns no values.

Syntax:

```
void Addition(int x )
 {
Statements;
Body;
 }
```

4) With Returntype - With Parameter: [Function with arguments and return value]

Methods can have a return type and parameters. The return type specifies the type of value the method will return, and parameters are the values passed to the method when it is called. Function with arguments and return value means both the calling function and called function will receive data from each other.

Syntax:

```
int Addition(int x )
 {
Statements;
Body;
 return x;
 }
```

****

Constructors in Java

Constructor is a block of codes similar to the method. It is a special type of method which is used to initialize the object. Constructor is invoked when an object of the class is created. Constructors must have the same name as the class. Constructors does not have any return type (not even void). The constructor is called when an object of a class is created. At the time of calling the constructor, memory for the object is allocated in the memory. It can be used to set initial values for object attributes. Constructors are called only once at the time of Object creation while method(s) can be called any number of times. A constructor can be overloaded but cannot be overridden.

A constructor in Java is a special type of method that is used to initialize an object. It is called when an instance of an object is created and memory is allocated for the object. Unlike Java methods, a constructor has the same name as that of the class and does not have any return type. Whenever an object of a class is created using a new keyword, it invokes a constructor of that class. Even if you haven't specified any constructor in the code, the Java compiler calls a default constructor. The default constructor is used to assign default states and values, such as 0, null, etc., to the object.

Need of Constructors in Java:

- Constructors cannot have a return type and must have the same name as the class.
- They don't require manual calling because they are automatically called when an object is created.
- Constructors are crucial for initializing objects with default or initial states.
- In other words, a constructor is a special type of method that is used to initialize instance variables in a class.
- The sole purpose of the constructor is to initialize the data fields of objects in the class.
- Java constructor can perform any action but specially designed to perform initializing actions, such as initializing the instance variables.
- A constructor within a class allows constructing the object of the class at runtime. It is invoked when an instance of a class is created using the new operator.
- Constructor overloading is possible but overriding is not possible.

Note: It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor (constructor with no-arguments) if your class doesn't have any.

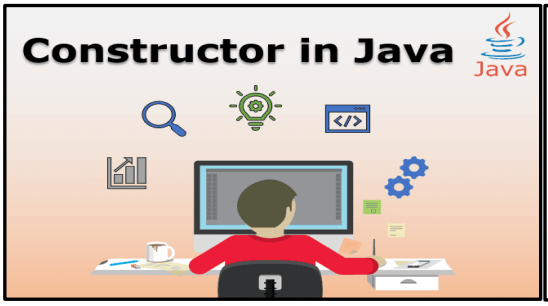
The rules for writing Constructors are as follows:

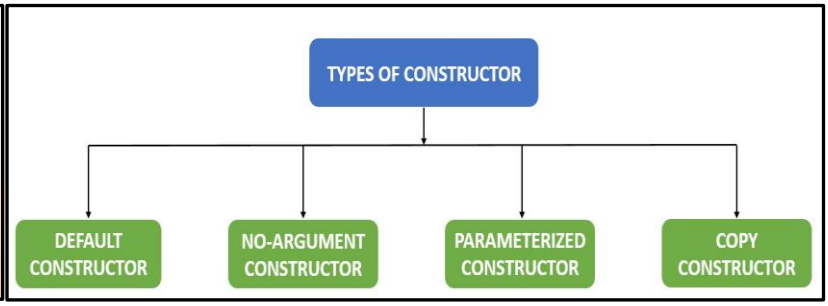
- ✓ The class name and constructor name should be the same.
- ✓ A constructor in Java cannot be abstract, final or static.
- ✓ It must have no explicit return type.
- ✓ A constructor can have an access modifier to control the access.
- ✓ Whenever we create an object/instance of a class, the constructor will be automatically called by the JVM (Java Virtual Machine).
- ✓ If we don't define any constructor inside the class, Java compiler automatically creates a default constructor at compile-time and assigns default values for all variables declared in the class.

The default values for variables are as follows:

- Numeric variables are set to 0.
- Strings are set to null.
- Boolean variables are set to false.

Syntax	Example
class ClassName { ClassName() //Constructor { // Constructor body } }	class Course { private String name; Course () { // constructor System.out.println("Constructor Called:"); name = "Java"; } public static void main(String[] args) { Main obj = new Main(); System.out.println("The name is " + obj.name); } }





1) No argument Constructors:

A constructor that has no parameter(arguments) is known as the No-argument or Zero argument constructor. If a constructor does not accept any parameters, it is known as a no-argument constructor. If you initialize multiple objects with a no-arg constructor, all the objects' values will be the same.

Syntax	Example
class ClassName { ClassName() // No argument Constructor { // Constructor body } }	**public class Person** { private String name; private int age; **public Person()** { // No argument Constructors this.name = "John Doe"; this.age = 30; System.out.println("Hello, my name is " + this.name + " and I am " + this.age + " years old."); } public static void main(String[] args) { // Main method Person john = new Person(); // Create an instance of the Person class } }

2) Parameterized Constructor:

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructors with parameters). You can use this type of constructor to assign different values to objects during creation. Parameterized constructors can be overloaded, allowing more than one version with different parameter lists to be called for various use cases. If we want to initialize fields of the class with our own values, then use a parameterized constructor. In addition, these types of constructors are usually used for constructor overloading to differentiate between multiple constructors with different datatypes.

Syntax	Example
class ClassName { ClassName (Parameters_List) // Paramerterized Constructor { // Constructor body } }	**class Person** { private String name; private int age; **public Person(String name, int age)** { // Paramerterized Constructor this.name = name; this.age = age; System.out.println("Name: " + name); System.out.println("Age: " + age); } } public class Main { public static void main(String[] args) { // Create instances of Person class Person person1 = new Person("John Doe", 30); Person person2 = new Person("Jane Doe", 25); } }

3) Default Constructor:

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor. A default constructor is invisible. Using these constructors, the instance variables of a method will be initialized with fixed values for all objects. The default constructor changed into the parameterized constructor. But Parameterized constructor can't change the default constructor.

Syntax	Example
class ClassName { **//No creation of any constructor by the user** public static void main (String[] args) { } }	**public class Product { //Creating a class 'Product'** String id; String product_name; double price; **//No creation of any constructor by the user** **public static void main (String[] args) {** Product product = new Product(); //Creating an object by calling //Trying to access the variables and print their default values System.out.println(product.id); System.out.println(product.product_name); System.out.println(product.price); } } **OUTPUT:** null null 0.0

4) Copy Constructor:

A copy constructor is used to create an exact copy of an object. It takes an object of the same class as a parameter and copies its attributes. A copy constructor allows you to do just that by defining a constructor that takes an existing object of the same class and copies its fields, providing a customized and flexible way to replicate objects. Although Java does not have a built-in copy constructor like some other languages (e.g., C++), you can create your own copy constructor by defining a constructor that takes an object of the same class as a parameter. This constructor copies the values of the fields from the provided object to the new object. This provides you with a level of control, as it's quite useful when you want to obtain a duplicate of an object with identical properties so that changes made in the newly created one will not affect the original one. Even though Java does not support any well-defined copy constructor, it can still be implemented manually.

Syntax	Example
class ClassName { **ClassName** **(Parameters_List)** // Paramerterized Constructor { // Constructor body } **ClassName (Constructor New Constructor)** // Copy constructor { // Copy Constructor body }	**class Car {** String model; int year; **Car (String model, int year) {** // Parameterized constructor this.model = model; this.year = year**;** **}** **Car (Car car) {** // Copy constructor this.model = car.model; this.year = car.year; **}** System.out.println("Model: " + model + ", Year: " + year); **public static void main(String[] args) {** Car car1 = new Car("Toyota", 2020); // Original object Car car2 = new Car(car1); // Copy constructor car2.displayInfo(); **} }** **OUTPUT:** Model: Toyota, Year: 2020

Difference Between Method and Constructor:

Method	Constructor
Method is used to expose the behaviour of an object.	Constructor is a special type of method that is used to initialize the state of an object.
Method can be executed when we explicitly call it	Constructor gets executed only when object is created.
Method name need not be same as the class name	Constructor name has to be same as the class name
Method has both void and return type.	Constructor does not have a return type even void also.
Method is not provided by the compiler in any case.	If we don't provide any constructor in the class, Java Compiler provides a default constructor for that class.
Method name may or may not be the same name as the class name.	Constructor name must be the same as name of the class.
The purpose of a method is to execute the functionality of the application.	The purpose of a constructor is to create an object of a class.
A method can be executed n number of times on an object.	Constructor will get executed only once per object.
Can have any access modifier (public, private, etc.)	Can have any access modifier (public, private, etc.)
They are inherited by subclasses.	They are not inherited by subclasses.

****

Inheritance in Java

The technique of creating a new class by using an existing class functionality is called inheritance in Java. In other words, inheritance is a process by which a child class acquires all the properties and behaviours of the parent class. The existing class is called parent class (a more general class) and the new class is called child class (a more specialized class). The child class inherits data and behaviour from the parent class.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. This promotes code reusability and establishes a natural hierarchy between classes.

Realtime Example of Inheritance in Java: In the real world, a child inherits the features of its parents such as beauty of mother and intelligence of father as shown in the below figure.

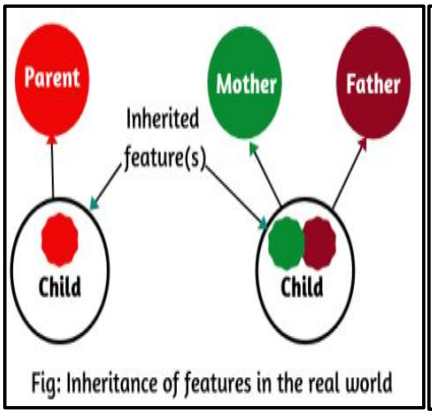
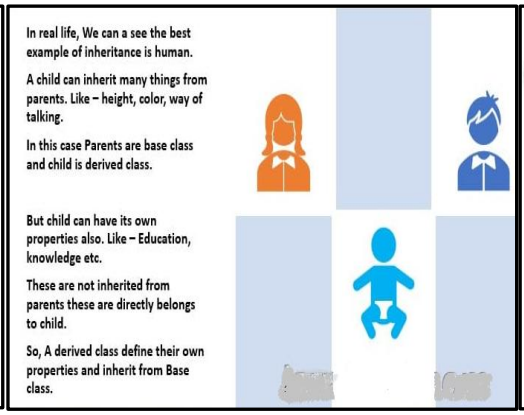
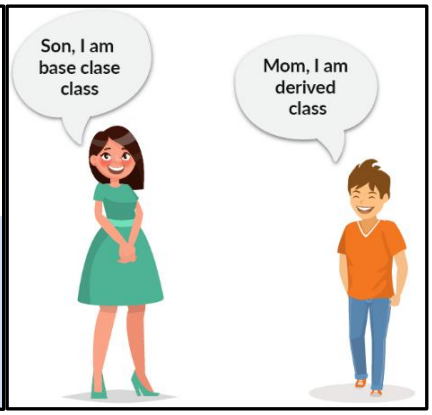


Fig: Inheritance of features in the real world





What is Is-A relationship in Java?

In Java the "Is-A" relationship refers to inheritance, where one class (the subclass or child class) inherits properties and behaviours (fields and methods) from another class (the superclass or parent class). This relationship is also known as a "parent-child" or "inheritance" relationship. For example, if a class named Mango inherits the class Fruit, then Mango has an "is-a" relationship with Fruit, which means that Mango is a Fruit. Is-a relationship, also known as the inheritance relationship, represents a type of relationship between two classes where one class is a specialized version of another. It implies that a subclass is a specific type of its superclass.

Is-A relationship in java represents Inheritance. It is implemented in Java through keywords extends (for class inheritance) and implements (for interface implementation). All the classes extends java.lang.object by default.

Let's take an example to understand Is-A relationship better. Look at the below figure to understand the Is-A relationship in java.

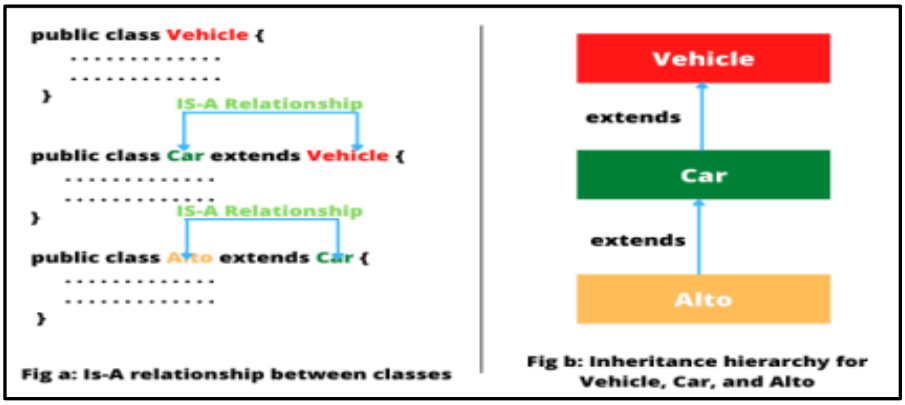
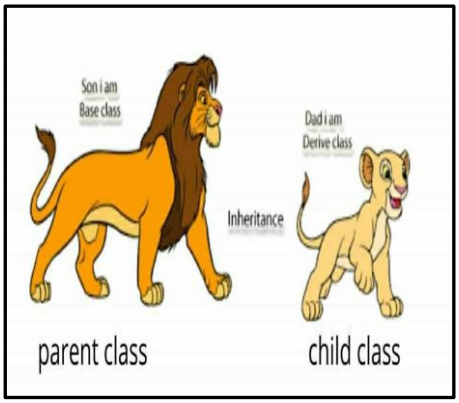


Fig a: Is-A relationship between classes

Fig b: Inheritance hierarchy for Vehicle, Car, and Alto



A car is a type of Vehicle, so inheritance hierarchy might begin from the Vehicle class as follows:

public class Vehicle { . . . . . }

public class Car extends Vehicle { . . . . . }

public class Alto extends Car { . . . . . }

We can say the following terms in inheritance relationship. They are as follows: Vehicle is the parent class (superclass) of Car.

Car is the child class (subclass) of Vehicle.

Car is the parent class of Alto.

Alto is the child class of Vehicle.

Car inherits from Vehicle.

Alto inherits from both Vehicle and Car.

Alto is derived from Car.

Car is derived from Vehicle.

Alto is derived from Vehicle.

Alto is a subtype of both Vehicle and Car.

In the terms of IS-A relationship, the following statements are true:

"Car extends Vehicle" means "Car IS-A Vehicle."

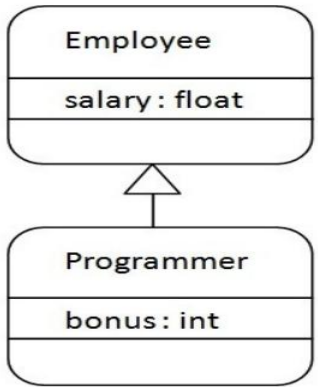
"Alto extends Car" means "Alto IS-A Car."

And we can also say that "Alto IS-A Vehicle".

Figure b shows the inheritance hierarchy for Vehicle, Car, and Alto. The arrows move from the child class to the parent class. In other words, the arrow of class points toward that class from which it extends (inherits).

Syntax	Syntax	Syntax
class baseclassname { list of variables list of methods } class derivedclassname **extends** baseclassname { list of variables list of methods }	class ParentClass { // ParentClass data variables // ParentClass member functions } class ChildClass extends ParentClass { // ChildClass data variables // ChildClass member functions }	class superclass { // superclass data variables // superclass member functions } class subclass extends superclass { // subclass data variables // subclass member functions }

Java Inheritance Example:




```
class Employee{    //ParentClass
float salary=40000; }
class Programmer extends Employee {  //ChildClass
int bonus=10000;
public static void main(String args[]){  //Main Method
Programmer p=new Programmer();  //Object
System.out.println("Programmer salary is:"+p.salary);
System.out.println("Bonus of Programmer is:"+p.bonus); } }
```

OutPut: Programmer salary is:40000.0
Bonus of programmer is:10000

Terms used in Inheritance:

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
Super Class/Parent Class/Base Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
Sub Class/Child Class/Derived Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
Extends Keyword: The extends keyword in Java is related to inheritance. The extends keyword extends a class (indicates that a class is inherited from another class). When a class needs to inherit the properties and behavior from another class, it has to use the extends keyword. The extends keyword is specified after the class name of the child class and before the class name of the parent class.
Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class

Why do we need/use Inheritance in Java?

- We can reuse the code from the base class.
- Using inheritance, we can increase features of class or method by Overriding.
- Inheritance is used to use the existing features of class.
- It is used to achieve runtime polymorphism i.e method overriding.
- Using inheritance, we can organize the information in a hierarchal form.

How is Inheritance implemented/achieved in Java:

Inheritance in Java is implemented through the use of the **extends** and **implements** keywords, which establish relationships between classes and interfaces. These relationships allow one class to inherit properties and behaviours from another class or to define behaviours that a class must implement.

extends keyword	implements keyword
The extends keyword in Java is used to indicate that a new class (called the subclass or derived class) is inheriting properties and behaviours (fields and methods) from an existing class (called the superclass or base class). This is a fundamental concept in object-oriented programming that facilitates code reuse, organization, and polymorphism.	The implements keyword in Java is used to indicate that a class is going to implement one or more interfaces. Interfaces in Java define a contract that a class must adhere to. They contain method signatures without implementations, and a class that implements an interface must provide concrete implementations for all of its methods.

Advantage of Inheritance in Java:

- One of the main advantages is that you can minimize the length of duplicate code in an application by putting the common code in the superclass and sharing amongst several subclasses.
- Due to reducing the length of code, redundancy of the application is also reduced.
- Inheritance can also make application code more flexible to change because a class that inherits from the superclass, can be used interchangeably.
- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- Inheritance provides a clear model structure which is easy to understand.
- In inheritance base class can decide to keep some data private so that it cannot be altered by the derived class.

Important Rules of Java Inheritance:

- We cannot assign a superclass to the subclass.
- We cannot extend the final class.
- A class cannot extend itself.
- One class can extend only a single class.
- Multiple Inheritance is NOT permitted in Java.
- Private members do NOT get inherited.
- Constructors cannot be Inherited in Java.
- In Java, we assign parent reference to child objects.

Behaviour of Access Modifiers in case of Java Inheritance:

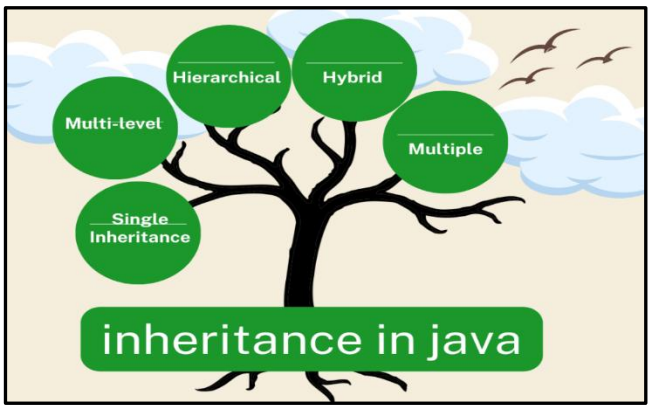
Superclass members can be inherited to subclass provided they are eligible by access modifiers. The behaviour of access specifiers in the case of inheritance in java is as follows:

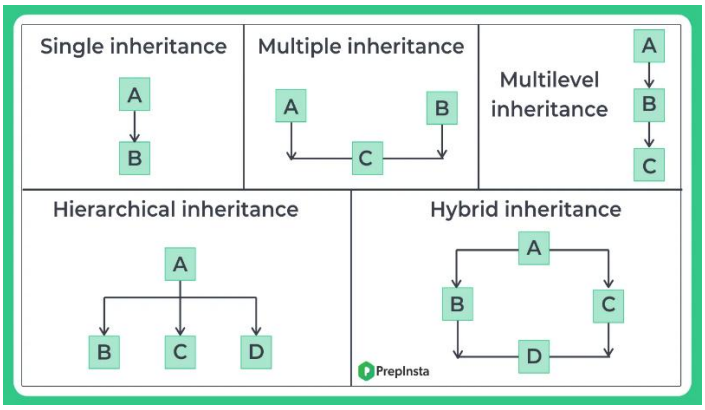
public	**public** members can be inherited to all subclasses.
private	The **private** members of the superclass cannot be inherited to the subclass because the private members of superclass are not available to the subclass directly. They are only available in their own class.
protected	The **protected** members of a parent class can be inherited to a derived class but the usage of protected members is limited within the package.
default	The **default** members of the parent class can be inherited to the derived class within the same package.

Access Modifier	Same Class	Same Package	Subclass (Same Package)	Subclass (Different Package)	Other Packages
public	Yes	Yes	Yes	Yes	Yes
private	Yes	No	No	No	No
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No

Types of Inheritance in Java:

Single-level inheritance	Multi-level Inheritance.	Hierarchical Inheritance.
Multiple Inheritance	Hybrid Inheritance.	

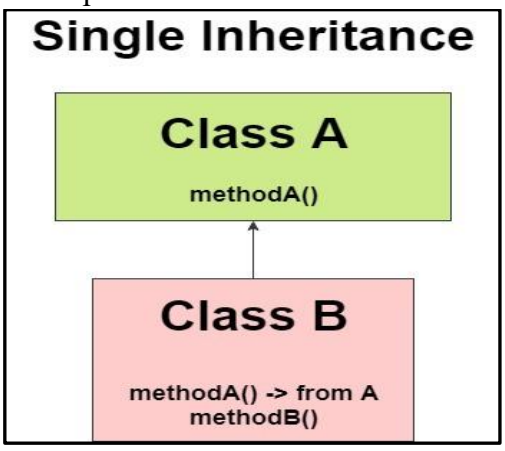


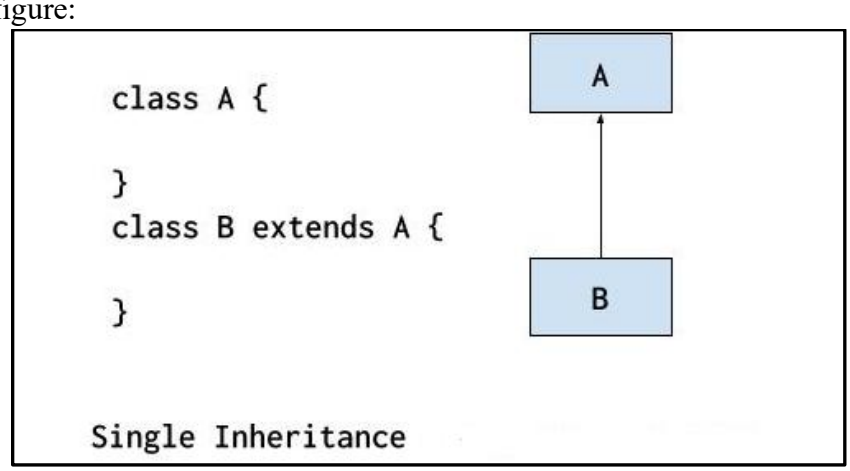


1) Single-Level Inheritance:

In single inheritance, subclasses inherit the features of one superclass. In other words, creating a subclass from a single superclass is called single inheritance. In single-level inheritance, there is only one base class and can be one derived class. The derived class inherits all the properties and behaviours only from a single class. Only one class is derived from the parent class. In this type of inheritance, the properties are derived

from a single parent class and not more than that. As the properties are derived from only a single base class the reusability of a code is facilitated along with the addition of new features.
It is represented as shown in the below figure:





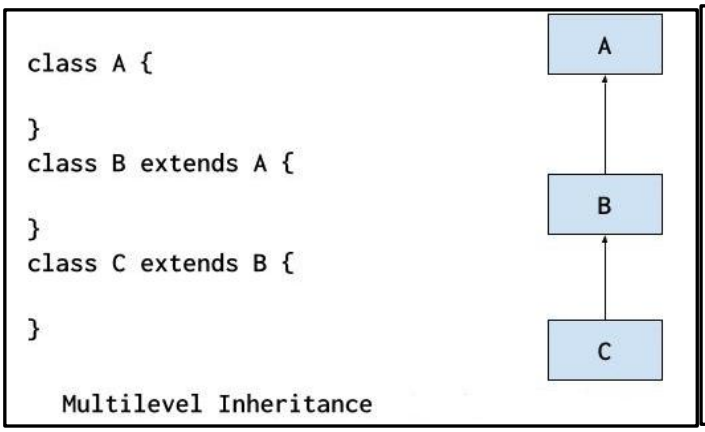
Two classes Class A and Class B are shown in Figure, where Class B inherits the properties of Class A.

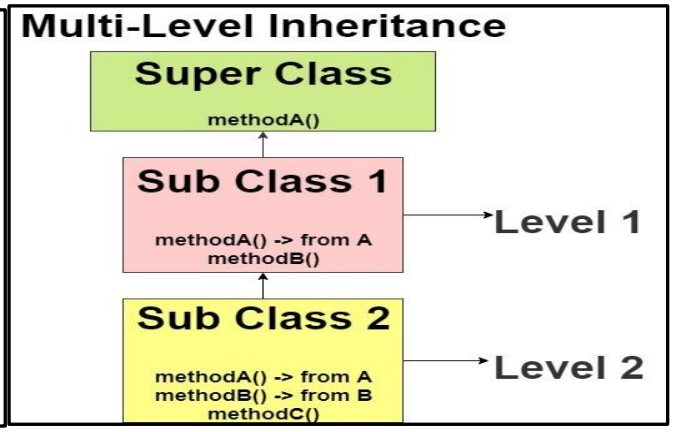
Syntax	Example
class A { method() { } } class B extends A { method() { } } class Test { main method() { method calling; parent and child methods; } }	class **Animal**{ void eat(){ System.out.println("eating..."); } } class **Dog** extends Animal{ void bark(){ System.out.println("barking..."); }} class **TestInheritance**{ public static void main(String args[]){ Dog d=new Dog(); d.bark(); d.eat(); }} **Output:** barking... eating...

2) Multi-Level Inheritance:

The multi-level inheritance includes the involvement of at least two or more than two classes. One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class. A class that is extended by a class and that class is extended by another class forming chain inheritance is called multilevel inheritance in java. In simple words, we can say that a class that has more than one parent class is called multi-level inheritance. In multilevel inheritance, there is one base class and one derived class at one level. At the next level, the derived class becomes the base class for the next derived class and so on. Note that the classes must be at different levels. Hence, there exists a single base class and single derived class but multiple intermediate base classes.

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.



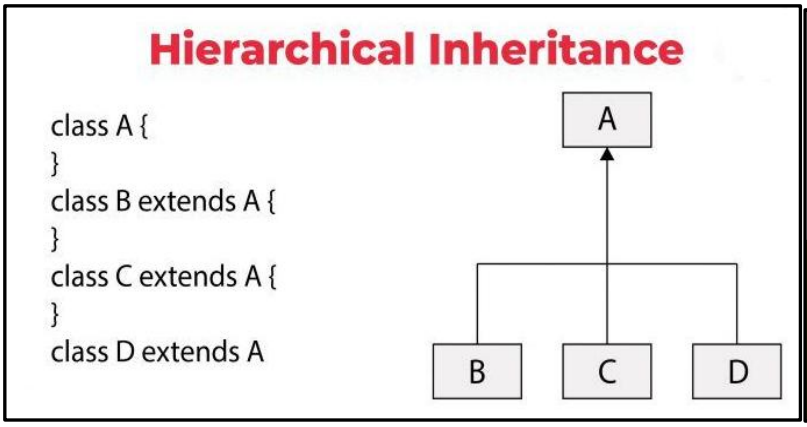


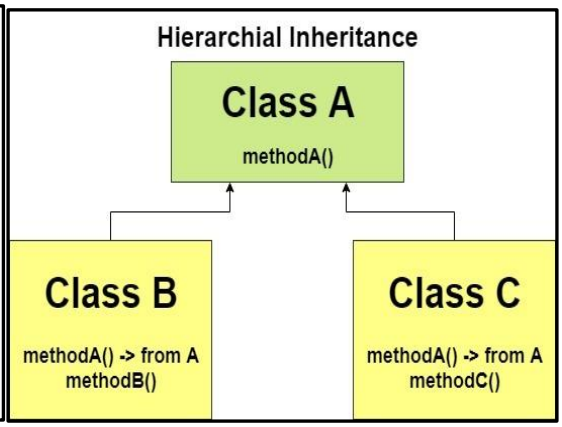
As you can see in the above diagram, class A is base class of class B (derived class), class B is the base class of class C (derived class).

Syntax	Example
class A{ method(){ body; } } class B extends A { method() { body; } } class C extends B { method() { body; } } class Test { main method() { method calling; parent and child methods; } }	class Animal{ void eat(){ System.out.println("eating..."); } } class Dog extends Animal{ void bark(){ System.out.println("barking..."); } } class BabyDog extends Dog{ void weep(){ System.out.println("weeping..."); } } class TestInheritance2{ public static void main(String args[]){ BabyDog d=new BabyDog(); d.weep(); d.bark(); d.eat(); }} **Output:** weeping... barking... eating...

3) Hierarchical Inheritance:

Many subclasses inherit from one single class is known as Hierarchical Inheritance in java. In other words, when one class is extended by many subclasses, it is known as hierarchical inheritance. Basically, it is a combination of more than one type of java inheritance. When a class contains several child classes or subclasses, or, to put it another way, when multiple child classes share the same parent class, this type of inheritance is referred to as hierarchical. Hierarchical Inheritance a combination of more than one type of inheritance. It is different from the multilevel inheritance, as the multiple classes are being derived from one superclass. These newly derived classes inherit the features, methods, etc, from this one superclass.





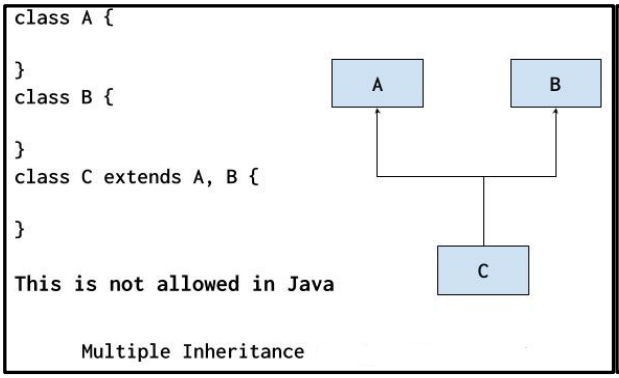
In the above diagram, class A is the parent (or base class) of all three classes B, C, and D. That is, classes B, C, and D inherit the same class A and can share all fields, methods of class A except private members. we can observe that the three classes Class B, Class C, and Class D are inherited from the single Class A. All the child classes have the same parent class in hierarchical inheritance.

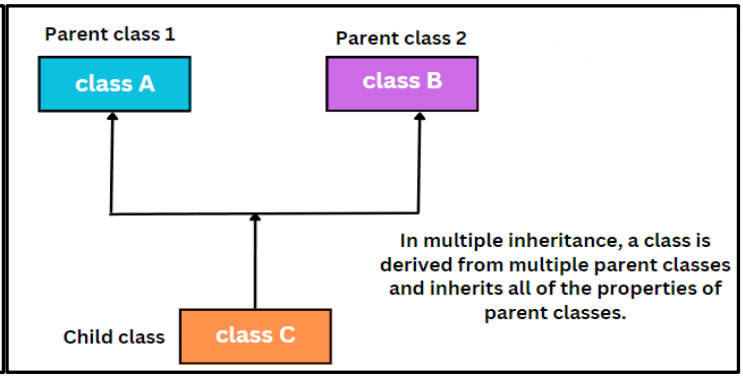
When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Syntax	Example
class A{ method(){ body; } } class B extends A { method() { body; } } class C extends A { method() { body; } } class Test { main method() { method calling; child methods; } }	class **Animal**{ void eat(){System.out.println("eating...");} } class **Dog** extends Animal{ void bark(){System.out.println("barking...");} } class **Cat** extends Animal{ void meow(){System.out.println("meowing...");} } class **TestInheritance3**{ public static void main(String args[]){ Cat c=new Cat(); c.meow(); c.eat(); //c.bark(); //C.T.Error }} **Output:** meowing... eating...

4) Multiple Inheritance:

Multiple Inheritances is a type of inheritance where a subclass can inherit features from more than one parent class. Multiple inheritances should not be confused with multi-level inheritance, in multiple inheritances the newly derived class can have more than one superclass. And this newly derived class can inherit the features from these superclasses it has inherited from, so there are no restrictions. In java, multiple inheritances can be achieved through interfaces. In other words, deriving subclasses from more than one superclass is known as multiple inheritance in java. In multiple inheritance, there can be more than one immediate super class and there can be one or more subclasses. Java does not support multiple inheritance through class.





Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. In other words, it can be described that subclass C inherits properties from both Class A and B.

Syntax	Example
class A{ method(){ body; } } class B { method() { body; } } class C extends A,B { method() { body; } } class Test { main method() { method calling; child methods; } }	class **A**{ void msg(){ System.out.println("Hello");} } class **B**{ void msg(){ System.out.println("Welcome");} } class **C** extends **A,B**{ //suppose if it were public static void main(String args[]){ C obj=new C(); obj.msg(); //Now which msg() method would be invoked? } } **Compile Time Error**

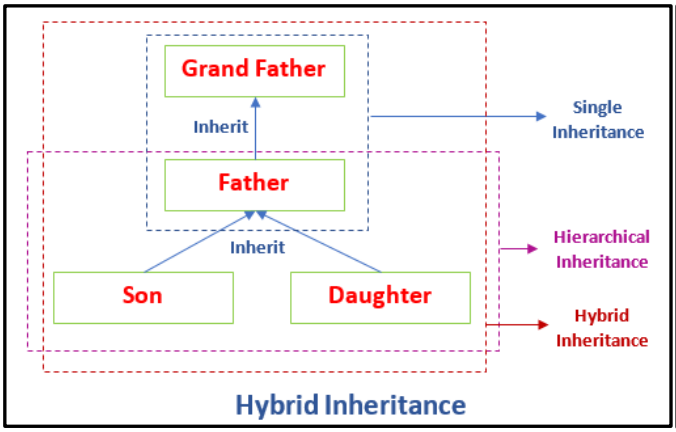
Why multiple inheritance is not supported in java?

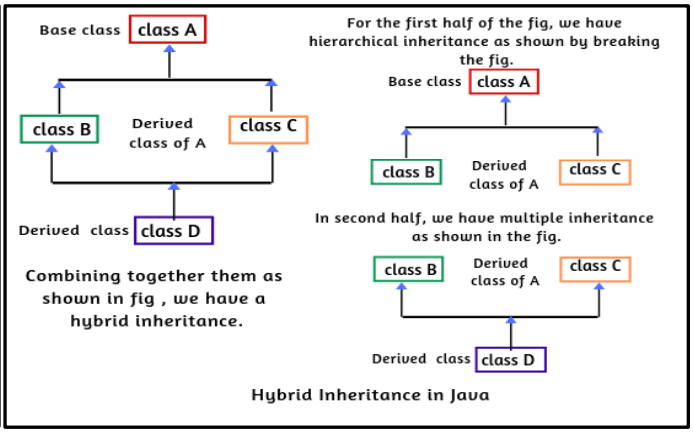
- To reduce the ambiguity, complexity, and confusion, Java does not support multiple inheritance through classes.
- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

5) Hybrid Inheritance:

Hybrid inheritance is a combination of more than two types of inheritances single and multiple. It can be achieved through interfaces only as multiple inheritance is not supported by Java. It is basically the combination of simple, multiple, hierarchical inheritances. Java, however, does not support hybrid inheritance directly due to its restriction on multiple inheritance with classes (i.e., a class cannot inherit from more than one class). This is to avoid ambiguity and complexity, such as the Diamond Problem.

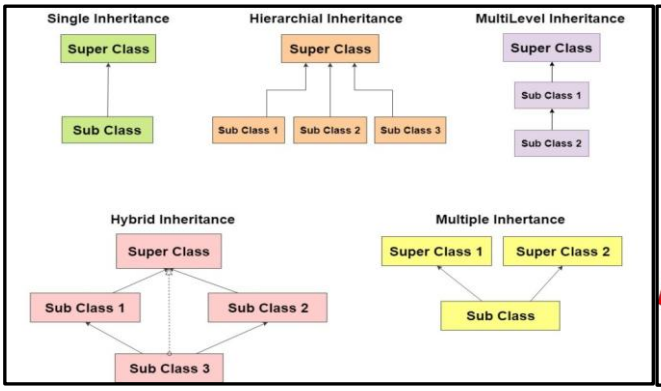
A hybrid inheritance is a combination of more than one types of inheritance. For example, there are four classes say A, B, C, and D. Assume that the class "A" and "B" extends the class "C". Also, another class, "D," extends the class "A". Here, the class "A" is a parent class for child class "D" and is also a child class for parent class "C".

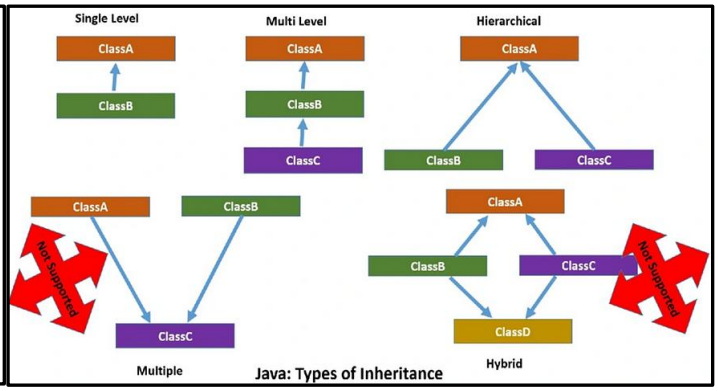




The hybrid inheritance can be achieved by using the following combinations:

- Single and Multiple Inheritance (not supported but can be achieved through interface)
- Multilevel and Hierarchical Inheritance
- Hierarchical and Single Inheritance
- Multiple and Multilevel Inheritance





****

Method OverLoading in Java

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Example: You have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. "In Java, two or more methods may have the same name these methods are called overloaded methods and this feature is called method overloading."

Method overloading is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding in Java. In Method overloading compared to parent argument, child argument will get the highest priority.

For Example:

```
void func() { ... }
void func(int a) { ... }
void func(int a, int b) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

Here, the func() method is overloaded. These methods have the same name but accept different arguments.

Note: The return types of the above methods are not the same. It is because method overloading is not associated with return types. Overloaded methods may have the same or different return types, but they must differ in parameters.

Why Method OverLoading?

Suppose, you have to perform the addition of given numbers but there can be any number of arguments (let's say either 2 or 3 arguments for simplicity).

In order to accomplish the task, you can create two methods sum2num (int, int) and sum3num (int, int, int) for two and three parameters respectively. However, other programmers, as well as you in the future may get confused as the behaviour of both methods are the same but they differ by name.

The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase the readability of the program.

Method OverLoading Rules in Java:

1.	The method name must be the same.
2.	Parameters must be different, i.e. each overloaded method must take a unique list of parameter types. We can change the parameters in one of the following three ways: ▪ Data type of parameters ▪ Number of parameters ▪ Sequence of data type of parameters
3.	Access modifiers can be anything or different.
4.	Return type can be anything or different.
5.	Exception thrown can be anything.
6.	The implementation does not matter in this case. A method can have any kind of logic.
7.	Method overloading is achieved by either: ▪ Changing the number of Parameters. ▪ Changing the data type of Parameters.

	▪ Changing the Sequence of Parameters.
Note	It is not method overloading if we only change the return type of methods. There must be differences in the number of parameters.

When to use Method OverLoading in Java	**Method OverLoading is done in java because of the following needs**	**Advantage of Method OverLoading**
▪ Method overloading is a powerful feature in Java but you should use as per needs. It should be used when you actually need multiple methods with different parameters, but methods do the same thing. If multiple methods perform different tasks, don't use the method overloading concept.	▪ Method overloading is done to reuse the same method name. ▪ It is done to make the program logically more readable and understandable. ▪ It is used to achieve the compile-time polymorphism in Java.	▪ Method overloading increases the readability of the program. ▪ In Java, Method overloading provides the same method name to reuse in the program. Method overloading helps to increases the readability and understanding of the program. ▪ It makes the program logically more readable and understandable. ▪ Java method overloading helps in achieving the compile-time polymorphism. ▪ It is useful to avoid repeating same code in different methods.

Different ways to Overload the Method in Java:

1) Method overloading by changing the number of arguments.
2) Method overloading by changing data type of parameters.
3) Method overloading by changing sequence of data type of parameters.

1) Method OverLoading by changing the number of Arguments (Parameters):

This involves defining multiple methods with the same name but a different number of parameters. Method overloading can be achieved by changing the number of parameters while passing to different methods. For example, if the 1 method of volume has 2 parameters and another method has 3 parameters, then it comes under Overloading on the basis of the number of parameters.

Syntax	**Example**
class ClassName{ method(datatype arg1) { //overload method1 body; } method(datatype arg1, datatype arg2) { //overload method2 body; } Main Method() { Object Creation; Method Calling; } }	**// Method overloading by changing the number of arguments** class AreaOverload { **// area of a circle** public float **area**(float radius) { return (float) (radius * radius * 3.14); } **//area of a rectangle** public float **area**(float length, float width) { return (length * width); } **public static void main(String args[]) {** AreaOverload a = new AreaOverload(); System.out.println("Area of rectangle: " + a.area(1.5f, 3.6f)); System.out.println("Area of circle: " + a.area(2.5f)); } } **Output:** Area of rectangle: 5.3999996 Area of circle: 19.625

Note: In this example, we have created two methods. The first **area** method performs finding the **area of a circle** of one parameter and the second **area** method performs on the **area of a rectangle** of two parameters.

2) Method OverLoading by changing DataType of Parameters:

The technique that allows you to define multiple methods within the same class with the same name but different parameters. This can include varying the number or type of parameters accepted by each method. For example, you can overload methods by changing the data type of their parameters. Having two or more methods (or functions) in a class with the same name and different arguments (or parameters).

Syntax	Example
class ClassName{ method(datatype arg1, datatype arg2) { //overload method1 body; } method(datatype arg1, datatype arg2) { //overload method2 body; } Main Method() { Object Creation; Method Calling; } }	**// Method overloading by changing the DataType of Parameters** public class Example { public int **sum**(int a, int b) { return a + b; } public double **sum**(double a, double b) { return a + b; } public static void **main(String[] args)** { **Example ex = new Example();** //Object int result1 = ex.sum(2, 3); double result2 = ex.sum(2.5, 3.5); System.out.println("Result1: " + result1); System.out.println("Result2: " + result2); } } **Output:** Result1: 5 Result2: 6.0
Note: In this example, we have created two methods. The first **sum** method performs finding the sum of two integer values and the second **sum** method performs sum of two double values.	

3) Method overloading by changing Sequence (Order) **of data type of Parameters:**

This involves defining multiple methods with the same name but different orders of parameters. Java identifies the correct method to call based on the order of parameters passed. The types of parameters remain the same, but their order in the method signature changes.

Syntax	Example
class ClassName{ method(datatype arg1, datatype arg2) { //overload method1 body; } method(datatype arg1, datatype arg2) { //overload method2 body; } Main Method() { Object Creation; Method Calling;	**//Method Overloading by Sequence**(Order)**of data type of Parameters** public class Changing_Sequence_Datatype { public void **details**(String name, char rank) { System.*out*.println("Employee name is "+name); System.*out*.println("Employee ranking is "+rank); } **//Method Overloading on sequence of parameters** public void **details**(char rank, String name { System.*out*.println("Employee ranking is "+rank); System.*out*.println("Employee name is "+name); } public static void **main(String[] args**) { **Changing_Sequence_Datatype emp = new Changing_Sequence_Datatype();** //Object emp.details("Rajesh", 'A'); // calls the first method (details(String, char)) emp.details('B', "Ram"); //calls the second method (details(char, String)) } }

} }	**Output:** Employee name is Rajesh Employee ranking is A Employee ranking is B Employee name is Ram
Note: In this example, we have created two methods. The first **details** method performs print the name and rank and the second **details** method performs print the rank and name.	

Method OverLoading Benefits:

- Method overloading significantly improves the readability of programs.
- The feature allows programmers to operate on tasks efficiently.
- Using Method Overloading results in codes that look neat and compact.
- Complex programs can be simplified using Method Overloading.
- Codes can be reused using this feature, which helps save memory as well.
- Since binding is accomplished in the compilation time, the execution period shortens.
- Method Overloading can also be used on constructors, which offers you opportunities to initialise objects of a class in different ways.
- Using Method Overloading, you can access the methods that perform related functions that use the same name, with a minor difference in the argument number or type.

Constructor Overloading in Java:

In Java, constructor overloading means to define multiple constructors but with different signatures. Constructor overloading is a technique of having more than one constructor in the same class with different parameter lists. In other words, defining two or more constructors with the same name in a class but with different signatures is called constructor overloading. We arrange them in a such a way that each constructor performs a different task. Java compiler differentiates these constructors based on the number of the parameter lists and their types. Therefore, the signature of each constructor must be different. The signature of a constructor consists of its name and sequence of parameter types. If two constructors of a class have the same signature, it represents ambiguity. Hence, overloaded constructors must have different signatures. The compiler decides which constructor has to be called, depending on the number of arguments passing with objects. By overloading constructors, you can provide flexibility and handle various scenarios when creating objects of a class. Constructor overloading is based on the concept of polymorphism.

Rules for Constructor OverLoading in Java:

Constructor Name	All constructors must have the same name as the class in which they are defined. This is a fundamental rule of constructors in Java.
Parameter Lists	Overloaded constructors must have different parameter lists. The parameter lists can differ in terms of the number, order, and data types of parameters.
Return Type	Constructors in Java do not have a return type, not even void. This rule applies to all constructors, including overloaded ones.
Access Modifiers	Constructors can have any access modifier (public, protected, default, or private), but they cannot be marked as static as they are instance methods.
Method Signature	The method signature for constructors includes the constructor's name and the parameter list. Overloaded constructors must have different method signatures to be distinguishable.

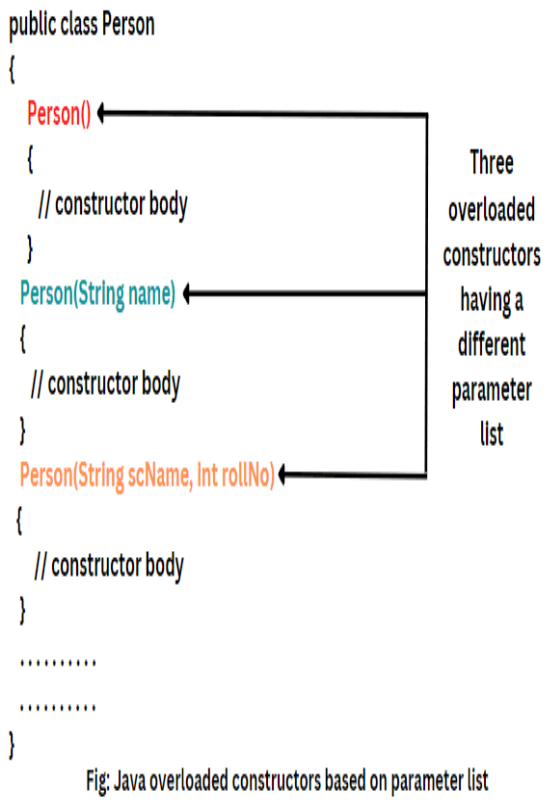
"The process of defining multiple constructors of the same class is referred to as Constructor overloading. However, each constructor should have a different signature or input parameters. In other words,

constructor overloading in Java is a technique that enables a single class to have more than one constructor that varies by the list of arguments passed. Each overloaded constructor is used to perform different task in the class."

Constructor Overloading Types in Java:

A. **No-Argument Constructor:** A no-argument constructor in Java is a constructor that takes no parameters. It's also known as a default constructor because it provides a default way to initialize an object when no specific values are provided during object creation.
B. **Constructor with Parameters:** A constructor with parameters in Java is a special method within a class that allows you to initialize object attributes using the values provided as arguments during object creation.
C. **Constructor Overloading with Different Parameter Types:** Constructor overloading with different parameter types involves creating multiple constructors in a class, each with a different set of parameter types.
D. **Constructor Overloading with Different Numbers of Parameters:** Constructor overloading with different numbers of parameters involves creating multiple constructors in a class, each with a different number of parameters.

Simple Constructor Overloading Example:

Syntax	Example
 Fig: Java overloaded constructors based on parameter list	public class Person { // **Declaring a non-parameterized constructor.** Person() { System.out.println("Introduction:"); } // **Declaring one parameterized constructor.** Person(String name){ System.out.println("Name: " +name); } // **Declaring two parameterized constructor**. Person(String scname, int rollNo) { System.out.println("School name: "+scname+ ", "+"Roll no:"+rollNo); } **public static void main(String[] args)** { // JVM will call constructor depending on arguments passed. Person p1 = new Person(); // calling with zero argument. Person p2 = new Person("John"); // calling with one argument. Person p3 = new Person("DPS", 12); // calling with two arguments. } } **Output:** Introduction: Name: John School name: DPS, Roll no:12

Difference between Constructor-Overloading and Method-Overloading in Java:

Constructor Overloading	Method Overloading
Constructor involves defining multiple constructors within a class, each with a different parameter list.	Method involves defining multiple methods within a class with the same name but different parameter lists.
Constructor cannot be invoked explicitly, invoked by new.	Method can be invoked by object reference.
They have the same name as the class and do not have a return type, not even 'void'.	They can have any anime except the class name and may have different return types or even no return type.
Constructors help in initializing objects when they are created.	Methods help in performing specific actions or tasks within a class.
A default constructor is provided if none is specified.	No default method is provided.
Constructor overloading is mostly utilized for initializing objects and offering many approaches to building class objects.	Multiple ways to execute an action on objects of a class are provided via method overloading.
Constructors are triggered when an object is formed using the new keyword.	The names of the methods are used to explicitly call them.

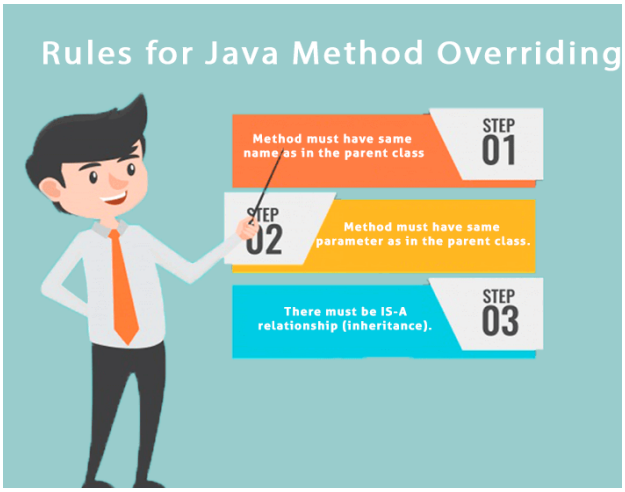
****

Method OverRiding in Java

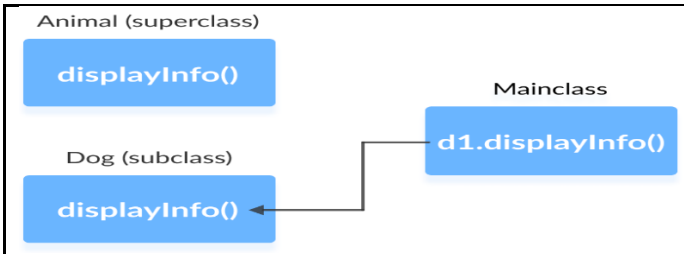
Method Overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The overridden method in the subclass should have the same signature (name, return type, and parameters) as the method in the superclass. In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Method overriding is one of the ways by which Java achieves Run Time Polymorphism. In Java, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and the same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Rules for Java Method Overriding:

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- The return type must be the same as the parent's method.
- There must be an IS-A relationship (inheritance).
- Final methods cannot be overridden.
- Static methods cannot be overridden.
- Private methods cannot be overridden.
- All the abstract methods in the parent class should be overridden in the child class.



In the last tutorial, we learned about inheritance. Inheritance is an OOP property that allows us to derive a new class (subclass) from an existing class (superclass). The subclass inherits the attributes and methods of the superclass. Now, if the same method is defined in both the superclass and the subclass, then the method of the subclass class overrides the method of the superclass. This is known as method overriding.



In the above program, the displayInfo() method is present in both the Animal superclass and the Dog subclass. When we call displayInfo() using the d1 object (object of the subclass), the method inside the subclass Dog is called. The displayInfo() method of the subclass overrides the same method of the superclass.

```
class Animal {
  public void displayInfo() {
    System.out.println("I am an animal.");    } }
class Dog extends Animal {
  @Override
  public void displayInfo() {
    System.out.println("I am a dog.");    } }
class Main {
  public static void main(String[] args) {
    Dog d1 = new Dog();
    d1.displayInfo();    } }
```

Output:

I am a dog.

Advantage of method overriding:

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class code. This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code. Overriding is when a child class has its method implementation for the method already

present in the parent class.

Difference between Method Overloading and Method Overriding:

Method OverLoading	Method OverRiding
Multiple methods with the same name but different parameters or types within the same class.	Providing a specific implementation for a method in the subclass that is already defined in the superclass.
Must have a different number or type of parameters.	Must have the same number and type of parameters.
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
Occurs in the same class.	Occurs between a superclass and its subclass.
Not related to inheritance.	Specifically involves inheritance.
Can overload static and final methods.	Cannot override static methods, and final methods cannot be overridden.
Method signature must be different, including the number or type of parameters	Method signature must be the same, including the number and type of parameters.
Private and final methods can be overloaded.	Private and final methods can't be overridden.
The return type can be the same or different	The return type must be the same or a subtype in the child class

****

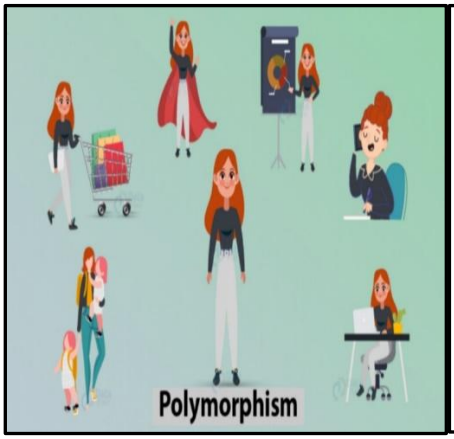
Polymorphism in Java

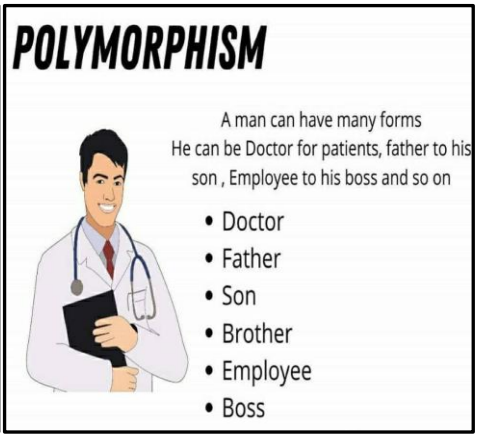
The word polymorphism is derived from two Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. Therefore, polymorphism means “many forms”. That is one thing that can take many forms. Polymorphism in Java is a concept by which we can perform a single action in different ways. That is, when a single entity behaves differently in different cases, it is called polymorphism in Java. Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. We can achieve flexibility in our code using polymorphism because we can perform various operations by using methods with the same names according to requirements. Polymorphism in java is one of the core concepts of object-oriented programming language (OOPs). We can perform polymorphism in java by method overloading and method overriding. Polymorphism is one of the 4 pillars of Object-Oriented Programming.

Realtime Example of Polymorphism in Java:

The best example of polymorphism is human behaviour. One person can have different behaviour. For example, a person acts as an employee in the office, a customer in the shopping mall, a passenger in bus/train, a student in school, and a son at home.

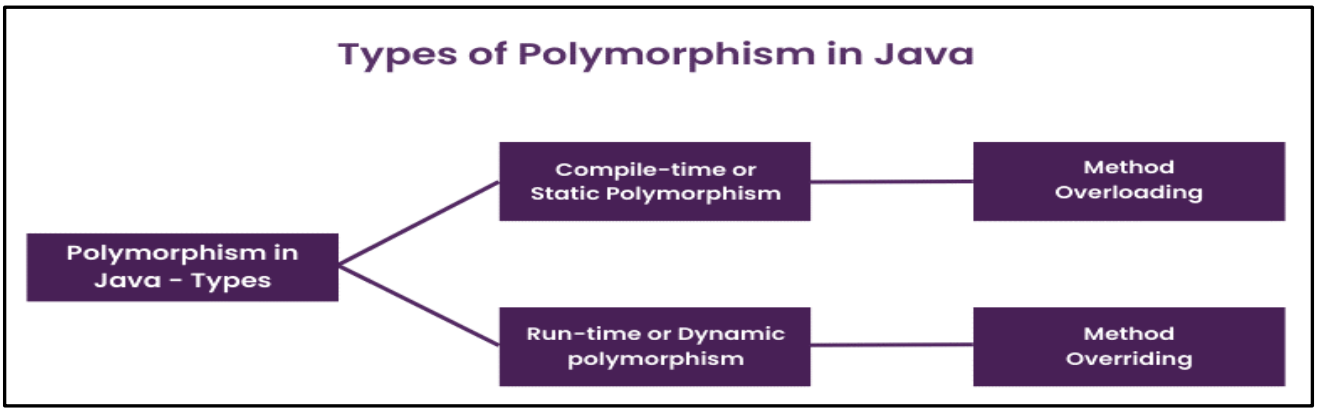
Another best example is your smartphone. The smartphone can act as a phone, camera, music player, alarm, and whatnot, taking different forms and hence polymorphism.





Types of Polymorphism in Java: There are two types of polymorphism in Java:

1) **Static polymorphism [Compile-time polymorphism]**
2) **Dynamic polymorphism [Runtime polymorphism]**



1) Static Polymorphism in Java (Compile Time polymorphism):

Static Polymorphism is also known as Compile-time polymorphism. A polymorphism that exhibited during compilation is called static polymorphism in java. In static polymorphism, the behaviour of a method is decided at compile-time. Hence, Java compiler binds method calls with method definition/body during compilation. Therefore, this type of polymorphism is also called compile-time polymorphism in Java and this type of polymorphism is achieved by function overloading or operator overloading. If you overload a

static method in Java, it is the example of compile time polymorphism.

Another example of static polymorphism is constructor overloading and method hiding. Compile-time polymorphism can be achieved/implemented by method overloading in java. Since binding is performed at compile-time. Static Polymorphism is also known as Static binding or Early binding and overloading as well.

Method Overloading: When there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments. It is one of the ways that Java implements polymorphism.

2) Dynamic Polymorphism in Java (Runtime Polymorphism):

Dynamic Polymorphism is also known as Compile-time polymorphism. A polymorphism that is exhibited at runtime is called dynamic polymorphism in java. In dynamic polymorphism, the behaviour of a method is decided at runtime, therefore, the JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called. This happens because objects are created at runtime and the method is called using an object of the class. The Java compiler has no awareness of the method to be called on an instance during compilation. Therefore, JVM invokes the relevant method during runtime. Dynamic or runtime polymorphism can be achieved/implemented in java using method overriding. Dynamic Polymorphism is also known as Dynamic binding or Late binding and overriding as well.

Method OverRiding is a mechanism where a method of Base class is overridden in the derived class to provide a more specific implementation. The signature of method in both base and derived classes is the same but they only differ in their implementation. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Difference between Static Polymorphism and Dynamic Polymorphism in Java:

Static Polymorphism	Dynamic Polymorphism
It can be defined as a process in which a function call with an object resolves at compile time by the compiler.	It can be defined as a process in which a function call with an object resolves at runtime.
Static Polymorphism also known as compile time polymorphism	Dynamic Polymorphism also known as runtime polymorphism
It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
It is analysed early at compile time so it provides face execution	It is slow because it is analysed at the runtime.
Inheritance is not involved.	Inheritance is involved.

Binding in Java:

If you have more than one method of the same name or two variables of the same name then there will be confusion during the compile-time and runtime of which method or variable is to be used as the reference code. This problem can be handled by the binding concept. Actually, a connection made between the method call and the method body is known as binding. The connecting (linking) between a method calls and method body/definition is called binding in java.

Types of Binding in Java:

There are two types of binding in java. They are as follows:

1. Static Binding (Also known as Early Binding).
2. Dynamic Binding (Also known as Late Binding).

Static Binding [Early Binding] in Java:

In static binding the method call is bonded with the method body at compile time. This is also known as early binding. This is done using static, private and, final methods. This binding is resolved at the compiled time by the compiler. In static binding, the java compiler does not check the type of object to which a particular reference variable is pointing to it. Java compiler just checks which method is going to be called through reference variable and method definition exists or not. This binding is also known as early binding because it takes place before the program actually runs. An example of static binding is method overloading.

Dynamic Binding [Late Binding] in Java:

In dynamic binding the method call is bonded with the method body at run time. This is also known as late binding. This is done using instance methods. In other words, the binding which occurs during runtime is called dynamic binding in java. This binding is resolved based on the type of object at runtime. In dynamic binding, the actual object is used for binding at runtime. Dynamic binding is also called late binding or runtime binding because binding occurs during runtime. An example of dynamic binding is method overriding.

Difference between Static and Dynamic Binding in Java:

▪ Static binding occurs at compile-time while dynamic binding occurs at runtime.
▪ In static binding, actual object is not used whereas, actual object is used in the dynamic binding.
▪ Static binding is resolved at compile time whereas, dynamic binding is resolved at runtime.
▪ **4.** Static binding is also called early binding because it happens during compilation whereas, dynamic binding is called late binding because it happens during runtime.
▪ **5.** An example of static binding is method overloading whereas, the example of dynamic binding is method overriding.
▪ **6.** Private, static, and final methods show static binding because they cannot be overridden whereas, except private, static, and final methods, other methods show dynamic binding because they can be overridden.

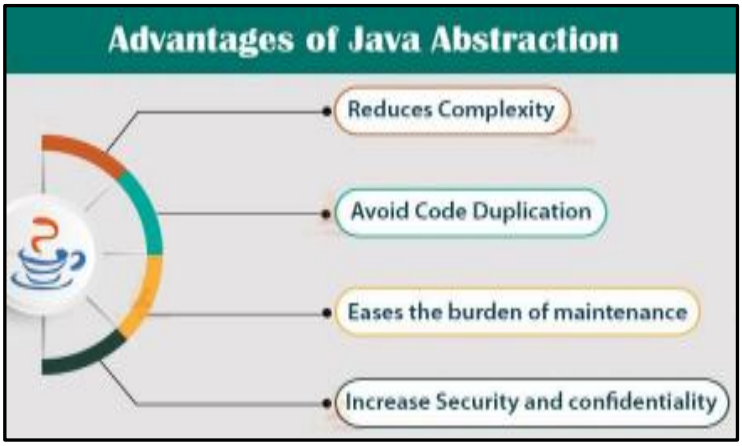
****

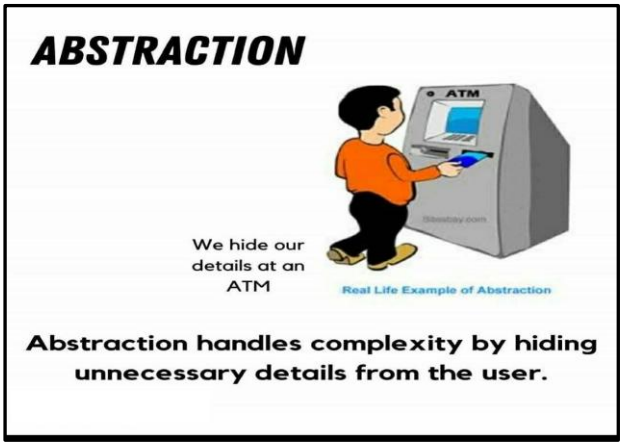
Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details. Abstraction refers to the practice of hiding implementation details and providing a simplified view of a system to its users. It is used to simplify complex systems by exposing only the necessary features and behaviours, while hiding the underlying complexity. Abstraction is important because it helps to simplify code, making it easier to use, maintain, and extend. By providing a simplified view of a system, abstraction helps to encapsulate complexity, making it easier to manage and work with. Abstraction in Java, which is one of the OOPs concepts, is the process of showing only the required information to the user by hiding other details.

Real-Life Example of Abstraction:

1. Consider a mobile phone, we just know that pressing on the call button, will dial the required number. But we actually don't know the actual implementation of how a call works. This is because the internal mechanism is not visible to us.
2. Sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
3. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.





Use of Abstraction:

Provides security by hiding implementation details.

Provides flexibility for the subclass to implement the functionality of the inherited abstract method of the superclass.

We can achieve abstraction in java by below 2 ways.

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java:

A class which is declared with the abstract keyword is known as an abstract class in Java. When we declare a class with an abstract keyword, we call it an abstract class. It can have abstract and non-abstract methods (method with the body). An abstract is a Java modifier applicable for classes and methods in Java but not for Variables. Java abstract class is a class that cannot be initiated by itself, it needs to be subclassed by another class to use its properties. Declaring a class as abstract means it can't be directly instantiated, which means an object can't be created from it. abstract is a non-access modifier keyword that we can use along with a class and method. An abstract class is a special class that is a superclass that contains unimplemented methods. Hence the subclass that inherits the parent abstract class should implement all the abstract methods.

Rules [Features] for Java abstract class:

- The class must have an abstract keyword in the class definition.
- The abstract class can have abstract methods and non-abstract methods.
- Java abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class must be declared abstract.3 It can have constructors either default or parameterize.
- Does not allow object creation for an abstract class.
- It allows defining final and static methods as well.
- We can access the abstract class only through inheritance by using the extends keyword.
- The class that extends the abstract class must implement all the abstract methods.
- An abstract class can include variables as well.

Syntax	Example
abstract class class_name { abstract return_type methodname(); //abstract method return_type methodname() //non-abstract method }	abstract class **Vehicle** { abstract void move(); } public class **Car** extends Vehicle { void move() { System.out.println("Inside the move method of the Car class..."); } public static void **main**(String args[]) { Car obj = new Car(); obj.move(); } } **Output:** Inside the move method of the Car class...

Abstract Methods in Java:

Abstract methods are those types of methods that don't require implementation for its declaration. These methods don't have a body which means no implementation. A method that doesn't have its body is known as an abstract method. We use the same abstract keyword to create abstract methods. When we use the keyword abstract while declaring a method, we call it an abstract method. Abstract methods have only a function declaration and do not have method implementation. This means it contains only an empty body and there is no code inside the method. The class that inherits the abstract class implements the abstract method. The abstract Method is used for creating blueprints for classes or interfaces. Here methods are defined but these methods don't provide the implementation. Abstract Methods can only be implemented using subclasses or classes that implement the interfaces. These methods are sometimes referred to as subclasses responsibility because they have no implementation specified in the super-class. Thus, a subclass must override them to provide a method definition.

Features of the Abstract Method:

- An abstract method in Java is declared through the keyword "abstract".
- While the declaration of the abstract method, the abstract keyword has to be placed before the name of the method.
- An abstract method contains a method signature, but no method body.
- An abstract method in Java doesn't have curly braces ({}), but the end of the method will have a semicolon (;)
- We can use abstract methods only inside an abstract class.
- The abstract method does not have any code within the method body.

Syntax	Example
abstract returntype methodname();	abstract class **Fruits** {

	abstract void taste(); //abstract method } class **Apple** extends Fruits { void taste() { System.out.println("Sweet taste"); } } public class AbstractDemo { public static void main(String[] args) { Apple a = new Apple(); a.taste(); }} **Output:** Inside the move method of the Car class...

****

Interface in Java

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is another mechanism to achieve abstraction. The interface in java enables java to support multiple-inheritance. An interface may extend only one interface, but a class may implement any number of interfaces. An interface is a completely "abstract class" that is used to group related methods with empty bodies. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables and constants. It cannot have a method body. Every interface in java is abstract by default. So, it is not compulsory to write abstract keyword with an interface. Once an interface is defined, we can create any number of separate classes and can provide their own implementation for all the abstract methods defined by an interface. A class that implements an interface is called implementation class. A class can implement any number of interfaces in Java. Every implementation class can have its own implementation for abstract methods specified in the interface as shown in the below figure.

Important Points for Interface:

- An interface is a container of abstract methods and static final variables.
- An interface, implemented by a class. (Class implements interface).
- An interface may extend another interface. (Interface extends Interface).
- An interface never implements another interface, or class.
- A class may implement any number of interfaces.
- We cannot instantiate an interface.
- Specifying the keyword abstract for interface methods is optional, it automatically added.

Why do we use Java interface?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
- Any class can extend only one class but can any class implement infinite number of interfaces.
- It can be used to achieve loose coupling.

Coupling in Java:

A situation where an object can be used by another object is termed as coupling. It is the process of collaborating together and working for each other. It simply means that one object requires another object to complete its assigned task. It is basically the usage of an object by another object, thereby reducing the dependency between the modules. It is called as collaboration if one class calls the logic of another class.

Types of Coupling:

1. **Tight Coupling** 2. **Loose Coupling**

1) Tight Coupling:

It is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its own class. The situation where an object creates another object for its usage, is termed as Tight Coupling. The parent object will be knowing more about the child object hence the two objects are called as tightly coupled. The dependency factor and the fact that the object cannot be changed by anybody else helps it to achieve the term, tightly coupled.

Example: Suppose you have made two classes. First class is a class called Volume, and the other class evaluates the volume of the box. Any changes that would be made in the Volume class, would be reflecting in the Box class. Hence, both the classes are interdependent on each other. This situation particularly is called as tight coupling.

2) Loose Coupling:

When an object gets the object to be used from external sources, we call it loose coupling. In other words,

the loose coupling means that the objects are independent. A loosely coupled code reduces maintenance and efforts. This was the disadvantage of tightly coupled code that was removed by the loosely coupled code. Let's take a look at some of the examples of loose coupling in Java.

How to declare an Interface?

Defining an interface is similar to that of a class. We use the keyword interface to define an interface. All the members of an interface are public by default. A class that implements an interface must implement all the methods declared in the interface. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. The following is the

Syntax for defining an Interface.

Syntax:	Example:	Converted code:
interface <interface_name> { // declare constant fields // declare methods that abstract // by default. }	interface HumanInterfaceExample { void learn (String str); void work (); int duration = 10; }	interface HumanInterfaceExample { public abstract void learn (String str); public abstract void work (); public static final int duration = 10; }
In the above code defines an interface HumanInterfaceExample that contains two abstract methods learn (), work () and one constant duration. Every interface in Java is auto-completed by the compiler. For example, in the above example code, no member is defined as public, but all are public automatically. The above code automatically converted as follows.		

Multiple Inheritance in Java by Interface:

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent classes to one child class. Multiple inheritance is a special type of inheritance in which a class can inherit properties of more than one parent class. It allows a class to have more than one superclass. If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance. As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Differences between Abstract class and Interface:

Abstract Class	Interface
An abstract class can be extended using the keyword "extends	The interface should be implemented using keyword implements
Can have public, protected, private and default modifier	Interface methods are by default public. you cannot use any other access modifier with it
It is faster than the interface	An Interface is somewhat slower & require extra indirection
An abstract class can have constructors	An interface cannot have constructors
An abstract class can extend another class and can implement multiple Java interfaces	The interface can extend another Java interface only

Difference between class and interface in Java:

Class	Interface
A class can be instantiated	An interface can never be instantiated
The class keyword is used to declare it	The interface keyword is used
The members of a class can be declared as private, public or protected	The members of an interface are always declared as public
Contains the concrete methods i.e methods with body	Contains abstract method i.e methods without the body
The extends keyword is used to inherit a class	The implements keyword is used to use an interface
Can contain final and static methods	Cannot contain final or static methods
A Java class can have constructors	An interface cannot have constructors
A class can extend only one class but can implement any number of interfaces	An interface can extend any number of interfaces but cannot implement any interface

****

Encapsulation in Java

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found. The process of binding data and corresponding methods (behaviour) together into a single unit is called encapsulation in Java. In other words, encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes. Thereby, we can keep variables and methods safes from outside interference and misuse. A combination of data hiding and abstraction is nothing but encapsulation.

Encapsulation = Data Hiding + Abstraction. If any component follows data hiding and abstraction, it is called an encapsulated component.

To achieve this, you must

- Declare class variables/attributes as private.
- Provide public get and set methods to access and update the value of a private variable. This approach restricts external access to specific attributes while still allowing them to be accessed by members of the current class via public getter and setter methods. You can specify which attributes can be read or updated using these methods and validate a new value before changing an attribute using them. In order to protect user data, encapsulation provides the fundamental property of hiding data.
- Every Java class is an example of encapsulation because we write everything within the class only those binds variables and methods together and hides their complexity from other classes.
- Another example of encapsulation is a capsule. Basically, capsule encapsulates several combinations of medicine.
- If combinations of medicine are variables and methods then the capsule will act as a class and the whole process is called Encapsulation.

Realtime Example of Encapsulation in Java:

Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, in this case, anyone can know your account balance. So, would you like it? Obviously, No. So, they declare balance variable as private for making your account safe, so that anyone cannot see your account balance. The person who has to see his account balance, will have to access only private members through methods defined inside that class and this method will ask your account holder name or user Id, and password for authentication. Thus, we can achieve security by utilizing the concept of data hiding. This is called Encapsulation in Java.

Syntax	Example
<Access_Modifier>class <Class_Name> { private <Data_Members>; private <Data_Methods>; }	public class Student { Private name; Private mobilenum; }

Advantage of Encapsulation in Java:

- The encapsulated code is more flexible and easier to change with new requirements.
- It prevents the other classes from accessing the private fields.
- Class attributes can be made read-only (if you only use the get method), or write-only (if you only use the set method)
- If you don’t define the setter method in the class, then the fields can be made read-only.
- If you don’t define the getter method in the class, then the fields can be made write-only.
- The programmer can change one part of the code without affecting other parts Increased security of data

Getter and Setter Method in Java:

Getter Method in Java:

A method which is used to retrieve/get the value of a variable or return the value of the private member variable is called getter method in Java. This method is also known as accessor method in Java. For every private variable, we should create a getter method. Depending on the access level giving to the variable, we can set the access modifier of its getter method. If we declare instance variables as private, we will have to add public getter methods for each one.

Syntax	Example
public returnType getName() { // Getter method's body. }	private String name; public String getName() { return name; }
In the above syntax, Name is the name of a variable and get is a keyword used to define a getter method in Java.	Here, we declared a variable name of type String with a private access modifier. private String name; Since the data type of variable name is String, the return type of getter method is also String.

Setter Method in Java:

A method which is used for updating or setting the value of a variable is called setter method in Java. This method is also known as mutator method in Java.

By using the setter method, we can modify or update the value of a variable. Just like with the getter method, we need to create a setter method for every variable in the class.

Syntax	Example
public void setName(formal_parameter) { // Setter method's body. }	private String name; public void setName(String name) { this.name = name; }
In the above syntax, Name is the name of variable and set is a keyword used to define setter method in Java.	

Note: Getter and Setter are methods used to protect your data and make your code more secure. Getter returns the value (accessors), it returns the value of data type int, String, double, float, etc.

Naming Convention for Defining Getter and Setter method in Java:

- If a variable (i.e. property) is of boolean type, the getter method's prefix can be either is or get.
- If a variable is not of boolean type, the getter method's prefix must be get.
- The prefix of a setter method must be set.
- The signature of a setter method must be public, with void as return type, and a formal parameter that represents the variable type or property type.
- The signature of a getter method must be public with no formal parameter, and a return type must match with the variable type.
- After prefix get or set, the first letter of variable name should be in the uppercase.

Differences between Data Hiding and Encapsulation in Java:

Data Hiding	Data Encapsulation
Data hiding can be considered as the parent process	Encapsulation is a sub-process of data hiding
Access specifier is always private	Access specifier can be private and public
Data hiding is about hiding method implementation	Encapsulation is about combining methods with data members
The main motto is to hide data and its implementation	The main motto is to combine data and their methods

****

Garbage Collection in Java

Garbage collection in Java is the process of automatically freeing heap memory by deleting unused objects that are no longer accessible in the program. Garbage collection is a process which is performed for memory management. It is used for reclaiming the runtime unused objects. In other simple words, the process of automatic reclamation of runtime unused memory is known as garbage collection. The program that performs garbage collection is called a garbage collector or simply a collector in java. It is a part of the Java platform and is one of the major features of the Java Programming language. Java garbage collector runs in the background in a low-priority thread and automatically cleans up heap memory by destroying unused objects. However, before destroying unused objects, it makes sure that the running program in its current state will never use them again. This way, it ensures that the program has no reference variable that does not refer to any object.

Dead object or Garbage in Java:

An object that cannot be used in the future by the running program is known as garbage in java. It is also known as dead object or unused object.

For example, an object exists in the heap memory, and it can be accessed only through a variable that holds references to that object.

What should be done with a reference variable that is not pointing to any object? Such situations can happen in the program. Consider the following two statements below:

```
Hello h1 = new Hello();
Hello h2 = new Hello();
h1 = h2;
```

Here, we have assigned one reference variable h1 to another reference variable h2. After the assignment statement h1 = h2, h1 refers to the same object referenced by h2 because the reference variable h2 is copied to variable h1. Due to which the reference to the previous object is gone. Thus, the object previously referenced by h1 is no longer in use i.e. the object referred by the reference variable h1 that is left side to the assignment operator, is not referring to the previous object and therefore is known as garbage or dead object.

In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing OutOfMemoryErrors. But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects. The main objective of Garbage Collector is to free heap memory by destroying unreachable objects. The garbage collector is the best example of the Daemon thread as it is always running in the background.

Advantages of Garbage Collection in Java:

- It makes java memory-efficient because the garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector (a part of JVM), so we don't need extra effort.

****

Exception Handling in Java

Exception handling in java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program can be maintained. In general, an exception means a problem or an abnormal condition that stops a computer program from processing information in a normal way. Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error). All the exceptions occur only at runtime. A syntax error occurs at compile time.

In other words, unwanted and unexpected behavior/event that interrupts the normal execution flow of the program is called exception in java. It is thrown from a method. The caller of the method can catch and handle the exception. An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. This exception must be handled to maintain the normal execution flow of the program.

If this exception is not handled suitably, the rest of code in the program will not be executed. To handle runtime exception, we use the exceptional handling technique in java programming. By handling the occurrence of exception, we can provide a meaningful message to the user about the error rather than a system-generated error message, which is difficult to understand for a user.

For example:

- If we access an array using an index that is out of bounds, we will get a runtime error named ArrayIndexOutOfBoundsException.
- If we enter a double value while the program expecting an integer value, we will get a runtime error called InputMismatchException.
- When JVM faces these kinds of errors or dividing an integer by zero in a program, it creates an exception object and throws it to inform us that an error has occurred. If we want to continue the execution of remaining code in the program, we will have to handle exception object thrown by error condition and then display a user-friendly message for taking corrective actions. This task is known as exception handling in java. If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.

Let's understand it with some real-time examples of Exception in Java

A. Suppose you are watching a video on YouTube, suddenly, internet connectivity is disconnected or not working. In this case, you are not able to continue watching the video on YouTube. This interruption is nothing but an exception.

B. Suppose a person is traveling by car from Hyderabad to Mumbai. After traveling mid distance, the tire of his car is punctured. This unexpected or unwanted event is nothing but an exception. The car owner always keeps an extra tire as an alternative on a long-distance journey. He changes the punctured tire with a new tire. After changing the tire, he continues the rest of the journey. This alternative way is called exception handling.

Why Exceptions occur in program:

- Opening a non-existing file in your program.
- Reading a file from a disk, but the file does not exist there.
- Writing data to a disk but the disk is full or unformatted.
- When the program asks for user input and the user enters invalid data.
- When a user attempts to divide an integer value by zero, an exception occurs.
- When a data stream is in an invalid format, etc.

Steps to solve Exception:

The purpose of exception handling is to detect and report an "exceptional circumstance" so that appropriate action can be taken. Error handling code performs the following tasks:

1. Find the problem (**Hit** the Exception).
2. Inform that an error has occurred (**Throw** the Exception).
3. Receive the error information (**Catch** the Exception).
4. Take corrective actions (**Handle** the Exception).

Error handling code consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions and take appropriate actions.

Advantage of Exception Handling:

a) The main advantage of exception handling technique is to maintain the normal flow of the program.
b) It provides flexibility in handling situations of errors.
c) It allows us to define a user-friendly message to handle the exception.
d) The exception handling technique helps to separate "Error-Handling code" from "Regular code."
e) The core advantage of exception handling is to maintain the normal flow of the application.
f) An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Let's consider a Scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.

Types of Java Exceptions:

Exceptions in java while programming is basically divided into two categories such as:

I. Built-in Exceptions:

These are the types of exception which can be caught using already existing java libraries. These exceptions are able to define the error situation so that we can understand the reason of getting this error. It can be categorized into two broad categories, i.e.

1.Checked Exception 2.Unchecked Exception

1.Checked Exception:

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer handles the exception or not. The programmer should have to handle the exception; otherwise, the system has shown a compilation error. The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.

For example: SQLException, IOException, InvocationTargetException, and ClassNotFoundException.

2.Unchecked Exception:

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these

exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error. Usually, it occurs when the user provides bad data during the interaction with the program. The classes that inherit the RuntimeException are known as unchecked exceptions

For Example: programming bugs like ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, Logical Errors, and using incorrect APIs.

II. User-Defined Exceptions:

In java we can create our own exception class and throw that exception using throw keyword. An Exception that is defined by the user/ programmer is known as a user-defined or custom exception. This exception occurs whenever there is some customizable or errors done by user while implementation and execution of program. You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes.

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

Exception	Description
NullPointerException	Exception is thrown when the JVM attempts to use an object reference that has not been initialized or is null. Common causes include calling methods on null objects, accessing fields of null objects, or attempting to use null as an array.
ArrayIndexOutOfBoundsException	Exception occurs when an application tries to access an array with an index that is outside the valid range (i.e., less than 0 or greater than or equal to the array's length).
ArithmeticException	Exception is thrown when an exceptional arithmetic condition has occurred. A common cause is division by zero.
ClassCastException	Exception is thrown when an application tries to cast an object to a subclass of which it is not an instance. For example, casting an Integer to a String.
IllegalArgumentException	Exception is thrown to indicate that a method has been passed an illegal or inappropriate argument. For example, passing a negative value where only positive values are expected.
IllegalStateException	Exception is thrown when a method is invoked at an illegal or inappropriate time. For example, calling next() on an iterator that has no more elements.
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string.
NumberFormatException	Exception is thrown when an attempt is made to convert a string to a numeric type, but the string does not have an appropriate format. For example, trying to parse "abc" into an integer.
UnsupportedOperationException	Exception is thrown to indicate that the requested operation is not supported by the object. For example, attempting to modify an unmodifiable collection.
FileNotFoundException	Exception is thrown when an attempt to open a file denoted by a specified pathname has failed. It indicates that the file does not exist.

EOFException (End Of File Exception)	Exception is thrown when an input operation has reached the end of a file or stream unexpectedly.
IOException	Exception is a general exception class that signals an I/O error has occurred. It is the superclass of all the exceptions that can occur during I/O operations.
SQLException	Exception is thrown when there is a database access error or other errors related to the database operations. It is part of the java.sql package.
NoSuchElementException	Exception is thrown when one tries to access an element that is not present. For example, calling next() on an empty iterator.
ConcurrentModificationException	Exception thrown when a collection is structurally modified while an iteration is in progress. For example, modifying a List while iterating over it.
OutOfMemoryError	Exception is a Error thrown when the JVM cannot allocate more memory. It can be due to memory leaks or insufficient heap space.
NoClassDefFoundError	Exception is thrown if the JVM cannot find the definition of a class that was present during the compilation but is not available at runtime.
ClassNotFoundException	Exception is thrown when an application tries to load a class through its name but cannot find it. It typically occurs with dynamic class loading using reflection.

****

Java Interview Questions and Answers

✓What is Java?

Java is a high-level programming language that was developed by James Gosling at Sun Microsystems in the mid-1990s. It is a platform-independent language that can be used to develop applications for desktop, mobile, and web platforms.

✓What are the features of Java?

Java has several features that make it a popular programming language, including Platform Independence, Object-Oriented, Simple, Secure, Robust, Multithreaded, Architecture Neutral, Portable, High Performance, Distributed, Dynamic, automatic memory management, multithreading, and robustness.

✓What is a class in Java?

A class is a blueprint or prototype from which objects are created. It defines a datatype by bundling data and methods that work on the data into one single unit. A class in Java is a blueprint which includes all your data. A class contains fields (variables) and methods to describe the behavior of an object. Let's have a look at the syntax of a class.

```
class Abc {
member variables // class body
methods}
```

✓What is an object in Java?

An object in Java is an instance of a class that contains data and behavior. It is created from a class using the "new" keyword.

An object has three characteristics:

State

Behavior

Identity

An object is created using the 'new' keyword. For example:

```
ClassName obj = new ClassName();
```

✓What is the difference between a class and an object?

A class is a blueprint or template for creating objects, while an object is an instance of a class.

✓Explain public static void main(String args[]) in Java.

In Java, we declared the main function as a public static void main (String args[]).

public: the public is the access modifier responsible for mentioning who can access the element or the method and what is the limit. It is responsible for making the main function globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.

static: static is a keyword used so that we can use the element without initiating the class so to avoid the unnecessary allocation of the memory.

void: void is a keyword and is used to specify that a method doesn't return anything. As the main function doesn't return anything we use void.

main: main represents that the function declared is the main function. It helps JVM to identify that the declared function is the main function.

String args[]: It stores Java command-line arguments and is an array of type java.lang.String class.

✓What is System.out and System.in ?

System.out – It is a PrintStream that is used for writing characters or can be said it can output the data we want to write on the Command Line Interface console/terminal.It will print to the standard out of the system. It is mostly used to display results on the console. It gives output on the console with the default(black) color. System.in – It is an InputStream used to read input from the terminal Window. We can't use the System.in directly so we use Scanner class for taking input with the system.in.

✓What is the platform?

A platform is the hardware or software environment in which a piece of software is executed. There are two types of platforms, software-based and hardware-based. Java provides the software-based platform.

✓Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation.

✓Can we execute a program without main() method?

No, It was possible before JDK 1.7 using the static block. Since JDK 1.7, it is not possible.

✓Why Java is not 100% Object-oriented?

Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

✓Name the Java IDE's?

Eclipse and NetBeans are the IDE's of JAVA.

✓What are JAR files?

Java Archive(JAR) is a file format that aggregates many files into a single file similar in a format similar to a ZIP file. In other words, the JAR is a zipped file comprised of java lass files, metadata, and resources like text, images, audio files, directories, .class files, and more.

Security – we can digitally sign JAR files

Compression – while using a JAR, we can compress files for efficient storage

Portability – we can use the same JAR file across multiple platforms

Versioning – JAR files can hold metadata about the files they contain

Sealing – we can seal a package within a JAR file. This means that all classes from one package must be included in the same JAR file

Extensions – we can use the JAR file format to package modules or extensions for existing software.

✓What is the JVM?

The Java Virtual Machine (JVM) is an abstract computing machine that enables a computer to run a Java program. It converts bytecode into machine code.

✓What is the JDK?

The Java Development Kit (JDK) is a software development kit used to develop Java applications. It includes the JRE, an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed for Java development.

✓What is the JRE?

The Java Runtime Environment (JRE) is a set of software tools for development of Java applications. It combines the Java Virtual Machine (JVM), platform core classes, and supporting libraries.

✓What is JIT?

JIT stands for (Just-in-Time) compiler is a part of JRE(Java Runtime Environment), it is used for better performance of the Java applications during run-time. JIT is a part of JVM, JIT is responsible for compiling bytecode into native machine code at run time.

✓What is the difference between a Local variable and an Instance/Global variable?

In Java, a local variable is typically used inside a method, constructor, or a block and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won't be even aware of that variable.

Instance/Global variable in Java, is a variable which is bounded to its object itself. These variables are declared within a class, but outside a method. Every object of that class will create it's own copy of the variable while using it. Thus, any changes made to the variable won't reflect in another instances of that class and will be bound to that particular instance only.

✓What are the data types supported in Java?

Primitive data types: byte, short, int, long, float, double, char, boolean.

Reference/Object data types: Strings, Arrays, Classes, Interfaces.

✓What is type casting? Difference between implicit and explicit casting?

Type casting is the process of converting one data type to another.

Implicit casting (widening): Automatically done by the compiler, converting a smaller type to a larger type. Example: int to long.

Explicit casting (narrowing): Manually done by the programmer, converting a larger type to a smaller type. Example: double to int, done using a cast operator: (int) myDouble.

✓What is the purpose of the final keyword when used with variables?

When the final keyword is used with a variable, it makes the variable's value constant. This means once a final variable is initialized, its value cannot be changed. Example: final int MAX = 100;.

final: variable: Its value cannot be changed.

final: method: Cannot be overridden.

final: class: Cannot be subclassed.

✓What are the data types in Java?

There are 2 types of data types in Java as mentioned below:

Primitive Data Type

Non-Primitive Data Type or Object Data type

Primitive Data Type: Primitive data are single values with no special capabilities. There are 8 primitive data types:

byte: stores an 8-bit signed two's complement integer

char: stores a single 16-bit Unicode character

short: stores a 16-bit signed two's complement integer

int: stores a 32-bit signed two's complement integer

long: stores a 64-bit two's complement integer

float: stores a single-precision 32-bit IEEE 754 floating-point

double: stores a double-precision 64-bit IEEE 754 floating-point

boolean: stores value true or false

Non-Primitive Data Type: Reference Data types will contain a memory address of the variable's values because it is not able to directly store the values in the memory. Types of Non-Primitive are mentioned below:

Strings

Array

Class

Object

Interface

✓What are the default values assigned to variables and instances in Java?

In Java When we haven't initialized the instance variables then the compiler initializes them with default values. The default values for instances and variables depend on their data types. Some common types of default data types are:

The default value for numeric types (byte, short, int, long, float, and double) is 0.

The default value for the boolean type is false.

The default value for object types (classes, interfaces, and arrays) is null.

The null character, "u0000, " is the default value for the char type.

✓What are comments in Java? Types of comments in Java?

Comments are annotations in the source code that are ignored by the compiler. They are used to make

the code more readable and to explain the code to other developers. Comments improve code readability, making it easier for others to understand, maintain, and extend the code. They are essential for collaborative development and code reviews.

There are three types of comments in Java:

Single-line comments: // comment

Multi-line comments: /* comment */

Documentation comments: /** comment */

✓What are the different types of errors in Java?

There are three main types of errors in Java:

Syntax errors (compile-time errors)

Runtime errors

Logical errors

Syntax errors occur when the code violates the syntax rules of the Java language, such as missing semicolons, incorrect use of keywords, or mismatched parentheses. These errors are detected at compile time.

Runtime errors occur during the execution of a program. They are usually caused by illegal operations, such as dividing by zero, accessing invalid array indices, or attempting to use null references.

Logical errors occur when the code compiles and runs without crashing, but the output is incorrect or does not meet the expected result due to flaws in the algorithm or logic used.

An Exception in Java is an event that disrupts the normal flow of the program's instructions. It is an object that is thrown at runtime when an error occurs.

✓What are the types of operators available in Java?

Arithmetic Operators (+, -, *, /, %)

Relational Operators (==, !=, >, <, >=, <=)

Logical Operators (&&, ||, !)

Bitwise Operators (&, |, ^, ~, <<, >>, >>>)

Assignment Operators (=, +=, -=, *=, /=, %=)

Unary Operators (+, -, ++, --)

Ternary Operator (? :)

✓How does the && operator differ from the & operator?

The && operator is a short-circuit logical AND operator. It only evaluates the right-hand side operand if the left-hand side operand is true. The & operator is a bitwise AND operator when used with integral types and a logical AND operator without short-circuiting when used with boolean types.

✓How do the bitwise shift operators <<, >>, and >>> work?

<< (left shift) shifts the bits of a number to the left, filling the rightmost bits with zeros.

>> (right shift) shifts the bits of a number to the right, preserving the sign bit (sign extension).

>>> (unsigned right shift) shifts the bits of a number to the right, filling the leftmost bits with zeros (zero extension).

✓What is the ternary operator in Java and how is it used?

The ternary operator (? :) is a shorthand for an if-else statement. It is used to evaluate a boolean expression and return one of two values based on the result. The syntax is: condition ? value1 : value2. If condition is true, value1 is returned; otherwise, value2 is returned.

Can you explain the difference between ++i and i++?

Both ++i (pre-increment) and i++ (post-increment) increment the value of i by 1. The difference is in their return value:

++i increments i and then returns the incremented value.

i++ returns the current value of i and then increments i.

✓What is operator precedence, and why is it important in Java?

Operator precedence determines the order in which operators are evaluated in an expression. Understanding operator precedence is important to predict the outcome of complex expressions correctly. For example, in the expression a + b * c, the multiplication (*) has higher precedence than addition (+), so b * c is evaluated first.

✓What is an infinite loop in Java? Explain with an example.

An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn't met. This type of loop can be the result of a programming error or may also be a deliberate action based on the application behavior. An infinite loop will terminate automatically once the application exits.

✓What is an Array in Programming?

An Array is a collection of similar data types stored in contiguous memory locations. An array in Java is a data structure that holds a fixed number of values of the same type. At the time of declaration of an array, you must specify the type of data with the array name.

You can access different elements present in an array using their index. For example, if you want to access an element present at the 3rd index(4th element) in an array arr, then you can write arr[3].

Declaration: int[] array;

Initialization: array = new int[10]; or int[] array = new int[]{1, 2, 3, 4, 5};

✓Mention some advantages and disadvantages of Arrays.

Advantages:

Multiple elements of Array can be sorted at the same time.

Using the index, we can access any element in O(1) time.

Disadvantages:

You need to specify how many elements you're going to store in your array ahead of time and We can not increase or decrease the size of the Array after creation.

You have to shift the other elements to fill in or close gaps, which takes worst-case O(n) time.

✓When will we get ArrayIndexOutOfBounds Exception?

ArrayIndexOutOfBoundsis a runtime exception that occurs when the program tries to access the invalid index of an array such as an Index higher than the size of the array or a negative index.

✓What is the default value of Array in Java?

If we don't specify the values by ourselves, then Java assigns default values in them which are 0 for byte, short, int, and long, 0.0 for float and double, false for boolean, and null for objects respectively.

On which memory arrays are created in Java?

Arrays in Java are created in heap memory. When an array is created with the help of a new keyword, memory is allocated in the heap to store the elements of the array. In Java, the heap memory is managed by the Java Virtual Machine(JVM) and it is also shared between all threads of the Java Program. The memory which is no longer in use by the program, JVM uses a garbage collector to reclaim the memory.

✓What are the types of an array?

Single-Dimensional Arrays: Arrays that have only one dimension i.e., an array of integers or an array of strings are known as single-dimensional arrays.

Multi-Dimensional Arrays: Arrays that have two or more dimensions such as two-dimensional or three-dimensional arrays.

✓What is the difference between int array[] and int[] array?

Both int array[] and int[] array are used to declare an array of integers in java. The only difference between them is on their syntax no functionality difference is present between them.

✓What is the String class in Java?

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

✓What is the difference between String, StringBuilder, and StringBuffer?

String is immutable, meaning its value cannot be changed once created. StringBuilder and StringBuffer are mutable classes used to create mutable (modifiable) string objects. StringBuilder is faster but not thread-safe, while StringBuffer is thread-safe.

✓What is Java StringPool?

A Java String Pool is a place in heap memory where all the strings defined in the program are stored. A separate place in a stack is there where the variable storing the string is stored. Whenever we create a new string object, JVM checks for the presence of the object in the String pool, If String is available in the pool, the same object reference is shared with the variable, else a new object is created.

✓How is the creation of a String using new() different from that of a literal?

String using new() is different from the literal as when we declare string it stores the elements inside the stack memory whereas when it is declared using new() it allocates a dynamic memory in the heap memory. The object gets created in the heap memory even if the same content object is present.

Syntax: String x = new String("ABC");

✓What is Object Oriented Programming?

Object-oriented programming, or OOP, is a programming model or approach in which programs are organized around objects rather than logic and functions. In other words, OOP mainly focuses on the objects that need to be manipulated instead of logic. This approach is ideal for large and complex programs and needs to be actively updated or maintained.

✓What are the main concepts of OOPs in Java?

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

Inheritance: Inheritance is a process where one class acquires the properties of another.

Encapsulation: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.

Abstraction: Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.

Polymorphism: Polymorphism is the ability of a variable, function or object to take multiple forms.

✓What is the difference between public, protected, default, and private access modifiers?

public: Accessible from any other class.

protected: Accessible within the same package and subclasses.

Default (no modifier): Accessible only within the same package.

private: Accessible only within the same class.

✓ What are the differences between the constructors and methods?

Java constructors are used for initializing objects. During creation, constructors are called to set attributes for objects apart from this few basic differences between them are:

Constructors are only called when the object is created but other methods can be called multiple times during the life of an object.

Constructors do not have a return type, whereas methods have a return type, which can be void or any other type.

Constructors are used to setting up the initial state but methods are used to perform specific actions.

✓ What is a copy constructor in Java?

Copy constructor is a member function that is used to initialize an object using another object of the same class. Though there is no need for copy constructor in Java since all objects are passed by reference. Moreover, Java does not even support automatic pass-by-value.

✓ What is inheritance in Java?

Inheritance is a mechanism wherein a new class is derived from an existing class. The new class inherits the properties and behaviors of the existing class.

- ✓ What is the 'IS-A ' relationship in OOPs Java?
  'IS-A' is a type of relationship in OOPs Java where one class inherits another class.
- ✓ What are the different types of inheritance in Java?
  Inheritance is the method by which the Child class can inherit the features of the Super or Parent class. In Java, Inheritance is of four types:
  Single Inheritance: When a child or subclass extends only one superclass, it is known to be single inheritance. Single-parent class properties are passed down to the child class.
  Multilevel Inheritance: When a child or subclass extends any other subclass a hierarchy of inheritance is created which is known as multilevel inheritance. In other words, one subclass becomes the parent class of another.
  Hierarchical Inheritance: When multiple subclasses derive from the same parent class is known as Hierarchical Inheritance. In other words, a class that has a single parent has many subclasses.
  Multiple Inheritance: When a child class inherits from multiple parent classes is known as Multiple Inheritance. In Java, it only supports multiple inheritance of interfaces, not classes.
  Note: Java doesn't support Multiple Inheritance.
- ✓ Can the constructor be inherited?
  No, we can't inherit a constructor.
  How can we restrict inheritance for a class?
  We can restrict inheritance for a class by the following steps:
  By using the final keyword
  If we make all methods final, then we cannot override that
  By using private constructors
  By using the Javadoc comment (" // ")
- ✓ What is method overloading?
  In Java, Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both. Method overloading in Java is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding. In Method overloading compared to the parent argument, the child argument will get the highest priority.
- ✓ What is method overriding?
  Method overriding is a method to achieve Run-time polymorphism in Java. Method overriding is a feature that allows a child class to provide a specific implementation of a method that is already provided by one of its parent classes. When a method in a child class has the same name, the same parameters or signature, and the same return type(or sub-type) as a method in its parent class, then the method in the subclass is said to override the method in the superclass.
- ✓ Can we override the private methods?
  It is not possible to override the private methods in Java. Method overriding is where the method in the subclass is implemented instead of the method from the parent class. The private methods are accessible only within the class in which it is declared. Since this method is not visible to other classes and cannot be accessed, it cannot be overridden.
- ✓ Can we override the static method?
  No, as static methods are part of the class rather than the object so we can't override them.
- ✓ What is polymorphism in Java?
  Polymorphism in Java allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.
- ✓ What is encapsulation?
  Encapsulation is the technique of making the fields in a class private and providing access to them via

public methods. It helps to protect the data from unauthorized access and modification. Data Encapsulation is the concept of OOPS properties and characteristics of the classes that The interface is binded together. Basically, it bundles data and methods that operate on that data within a single unit. Encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class.

- ✓ What are the advantages of Encapsulation in Java?

  The advantages of Encapsulation in Java are mentioned below:

  Data Hiding: it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class.

  Increased Flexibility: We can make the variables of the class read-only or write-only depending on our requirements.

  Reusability: Encapsulation also improves the re-usability and is easy to change with new requirements.

  Testing code is easy: Code is made easy to test for unit testing.
- ✓ What is Abstraction?

  Abstraction refers to the act of representing essential features without including background details. The detailed information or the implementation is hidden. The most common example of abstraction is a car, we know how to turn on the engine, accelerate and move, however, the way engine works, and its internal components are complex logic hidden from the general users. This is usually done to handle the complexity.
- ✓ What is an abstract class?

  An abstract class is a class that cannot be instantiated on its own and is designed to be subclassed. It may contain abstract methods, which are methods declared without an implementation.
- ✓ When Abstract methods are used?

  An abstract method is used when we want to use a method but want to child classes to decide the implementation in that case we use Abstract methods with the parent classes.
- ✓ What is an interface?

  An interface in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors.
- ✓ What is the difference between an abstract class and an interface?

  An abstract class can have instance methods that implement a default behavior. Interfaces can only declare methods, and default methods, but cannot provide implementation. A class can implement multiple interfaces but can extend only one class.
- ✓ What is a package in Java?

  A package is a namespace that organizes a set of related classes and interfaces. Packages are used to avoid name conflicts and to control access to classes, interfaces, and other components.

  There are two types of packages in Java

  User-defined packages

  Build In packages
- ✓ Can we declare Pointer in Java?

  No, Java doesn't provide the support of Pointer. As Java needed to be more secure because which feature of the pointer is not provided in Java.
- ✓ What is exception handling in Java?

  Exception handling in Java is a mechanism to handle runtime errors, allowing the program to continue its normal flow. It is done using five keywords: try, catch, finally, throw, and throws.
- ✓ What is the difference between checked and unchecked exceptions?

Checked Exception:

Checked Exceptions are the exceptions that are checked during compile time of a program. In a program, if some code within a method throws a checked exception, then the method must either handle the exception or must specify the exception using the throws keyword.

Checked exceptions are of two types:

Fully checked exceptions: all its child classes are also checked, like IOException, and InterruptedException.

Partially checked exceptions: some of its child classes are unchecked, like an Exception.

Unchecked Exception:

Unchecked are the exceptions that are not checked at compile time of a program. Exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

- ✓ What is the base class for Error and Exception?

Error is an illegal operation performed by the user which causes abnormality in the program. Exceptions are the unexpected events or conditions that comes while running the program, exception disrupts the normal flow of the program's instructions.

Errors and Exceptions both have a common parent class which is java.lang.Throwable class.

- ✓ What are the benefits of exception handling?

Exception handling helps to maintain the normal flow of the application, separates error-handling code from regular code, provides a method for reporting error conditions, and allows handling multiple exceptions.

NullPointerException is thrown when the JVM attempts to use an object reference that has not been initialized or is null. Common causes include calling methods on null objects, accessing fields of null objects, or attempting to use null as an array.

ArrayIndexOutOfBoundsException occurs when an application tries to access an array with an index that is outside the valid range (i.e., less than 0 or greater than or equal to the array's length).

ArithmeticException is thrown when an exceptional arithmetic condition has occurred. A common cause is division by zero.

ClassCastException is thrown when an application tries to cast an object to a subclass of which it is not an instance. For example, casting an Integer to a String.

IllegalArgumentException is thrown to indicate that a method has been passed an illegal or inappropriate argument. For example, passing a negative value where only positive values are expected.

IllegalStateException is thrown when a method is invoked at an illegal or inappropriate time. For example, calling next() on an iterator that has no more elements.

IndexOutOfBoundsException is a superclass of exceptions that are thrown when an index is out of range for arrays, strings, or collections. It includes ArrayIndexOutOfBoundsException and StringIndexOutOfBoundsException.

NumberFormatException is thrown when an attempt is made to convert a string to a numeric type, but the string does not have an appropriate format. For example, trying to parse "abc" into an integer.

UnsupportedOperationException is thrown to indicate that the requested operation is not supported by the object. For example, attempting to modify an unmodifiable collection.

FileNotFoundException is thrown when an attempt to open a file denoted by a specified pathname has failed. It indicates that the file does not exist.

EOFException (End Of File Exception) is thrown when an input operation has reached the end of a file or stream unexpectedly.

IOException is a general exception class that signals an I/O error has occurred. It is the superclass of all the exceptions that can occur during I/O operations.

SQLException is thrown when there is a database access error or other errors related to the database

operations. It is part of the java.sql package.

NoSuchElementException is thrown when one tries to access an element that is not present. For example, calling next() on an empty iterator.

ConcurrentModificationException is thrown when a collection is structurally modified while an iteration is in progress. For example, modifying a List while iterating over it.

OutOfMemoryError is a Error thrown when the JVM cannot allocate more memory. It can be due to memory leaks or insufficient heap space.

NoClassDefFoundError is thrown if the JVM cannot find the definition of a class that was present during the compilation but is not available at runtime.

ClassNotFoundException is thrown when an application tries to load a class through its name but cannot find it. It typically occurs with dynamic class loading using reflection.

- ✓ What purpose do the keywords final, finally, and finalize fulfill?

  final: final is a keyword is used with the variable, method, or class so that they can't be overridden.

  finally: finally is a block of code used with "try-catch" in exception handling. Code written in finally block runs despite the fact exception is thrown or not.

  finalize: It is a method that is called just before deleting/destructing the objects which are eligible for Garbage collection to perform clean-up activity.
- ✓ What is garbage collection in Java?

  Garbage collection in Java is the process of reclaiming the runtime unused memory automatically. It is a form of automatic memory management.
- ✓ Why Garbage Collection is necessary in Java?

  For Java, Garbage collection is necessary to avoid memory leaks which can cause the program to crash and become unstable. There is no way to avoid garbage collection in Java. Unlike C++, Garbage collection in Java helps programmers to focus on the development of the application instead of managing memory resources and worrying about memory leakage. Java Virtual Machine (JVM) automatically manages the memory periodically by running a garbage collector which frees up the unused memory in the application. Garbage collection makes Java memory efficient because it removes unreferenced objects from the heap memory.
- ✓ What is the difference between abstract class and interface in Java 8?

  Abstract Class: Can have abstract and non-abstract methods.

  Interface: Can have default and static methods in addition to abstract methods.
- ✓ What is the difference between an abstract class and an interface?

  An abstract class is a class that cannot be instantiated and may contain both abstract and non-abstract methods, while an interface is a collection of abstract methods that must be implemented by any class that implements it.
- ✓ What is the Java Virtual Machine (JVM)?

  The JVM is an abstract computing machine that provides the runtime environment for Java programs. It interprets compiled Java code and executes it on the computer hardware.
- ✓ What is a package in Java?

  A package in Java is a mechanism for organizing classes and interfaces into namespaces. It helps to prevent naming conflicts and makes it easier to locate and use classes in a large project.
- ✓ What is the difference between public, private, and protected access modifiers in Java?

  Public, private, and protected are access modifiers in Java that determine the accessibility of class members. Public members can be accessed from anywhere, private members can only be accessed within the class, and protected members can be accessed within the class and its subclasses.
- ✓ What is a static method in Java? Java Interview Questions and Answers

  A static method in Java is a method that belongs to the class rather than to any specific instance of the

class. It can be called using the class name, rather than an instance of the class.

- ✓ What is a final keyword in Java?
  The final keyword in Java can be used to indicate that a class cannot be extended, a method cannot be overridden, or a variable cannot be reassigned. It is often used to define constants.
- ✓ What is a thread in Java?
  A thread in Java is a lightweight process that allows a program to execute multiple tasks concurrently. Each thread has its own call stack, but shares the same memory and resources as other threads in the program.
- ✓ What is a try-catch block in Java?
  A try-catch block in Java is used for exception handling. The code that may throw an exception is placed in the try block, and any exceptions that are thrown are caught and handled in the catch block.
- ✓ What is the difference between checked and unchecked exceptions in Java?
  Checked exceptions are checked at compile-time and must be handled by the programmer, while unchecked exceptions are not checked at compile-time and can be handled at runtime if necessary.
- ✓ What is the difference between a private and a protected access modifier in Java?
  A private access modifier in Java restricts access to the member to only within the same class, (https://fullstackadda.com/) while a protected access modifier allows access to the member within the same class and its subclasses.
- ✓ What is the difference between a static and a non-static method in Java?
  A static method in Java is a method that belongs to the class and can be called without an instance of the class, while a non-static method is a method that belongs to an instance of the class.
- ✓ What is a package in Java? What is the purpose of a package in Java?
  A package in Java is a namespace that organizes a set of related classes and interfaces.
  A package in Java is a way of organizing related classes and interfaces into a single unit, making it easier to manage and maintain large code bases. Packages also help to avoid naming conflicts between classes and provide a level of access control to the classes within them.
- ✓ What is the difference between a private and a protected method in Java?
  A private method can only be accessed within the same class, while a protected method can be accessed within the same class and any subclasses. Java Interview Questions and Answers
- ✓ How is memory allocation managed in Java?
  Memory allocation in Java is managed by the JVM, which allocates memory on the heap for objects and on the stack for local variables and method calls. Java Interview Questions and Answers
- ✓ What is the purpose of the “this” keyword in Java?
  The “this” keyword in Java refers to the current object and is used to refer to instance variables and methods within the class.

****