What is a Programming Language -

A programming language is a formal language that is used to write instructions or programs that can be executed by a computer. It is a set of rules, symbols, and syntax that allow programmers to express algorithms and computations in a way that the computer can understand and execute.

Types of Programming Languages -

➢ **Structured Programming Language -** Structured programming language is a type of programming language that emphasises the use of structured control flow constructs such as loops, conditionals, and subroutines to improve the clarity, reliability, and maintainability of software to make programs easier to read and maintain and don't support any OOPs concepts.
Example - C, Python, etc…

➢ **Object Based programming Language -** Object-based programming languages are languages that support objects, but do not support all of the features of object-oriented programming (OOP) languages. Object-based languages allow for the creation of objects, which are instances of classes, but may not support features such as inheritance and polymorphism.
Example - Java Script, VB Script, Python, etc…

➢ **Object Oriented Programming Language -** Object-oriented programming (OOP) language is a type of programming language that is based on the concept of objects. OOP languages allow developers to define objects that can contain both data and behaviour, making it easier to organise and manage complex code.
Example - C++, C#, Java, Python, etc…

What is Java -

Java is a high-level, general-purpose, object-oriented, and secure programming language.

History of Java -

Developed by 'James Gosling' and his team in the mid-1990s in Sun Microsystems. Later Java was acquired by Oracle Corporation in 2009.
Java was firstly named as 'Oak', which was changed in 1995 to 'Java'.

Platform-

Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API(Application Programming Interface), it is called platform.

Flavours of Java -

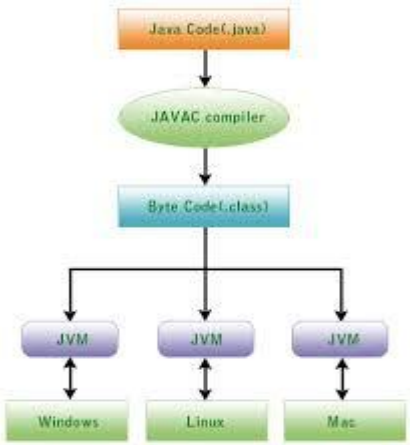
➢ **Core Java(J2SE) -** Core Java refers to the fundamental concepts and basic syntax of the Java language.

➢ **Advanced Java(J2EE) -** Advanced Java refers to the advanced topics, specialised libraries and frameworks that extend the functionality of the Java language.

➢ **Android Java(J2ME) -** Android Java refers to the Java programming language used for developing mobile applications for the Android platform.

Features Of Java –

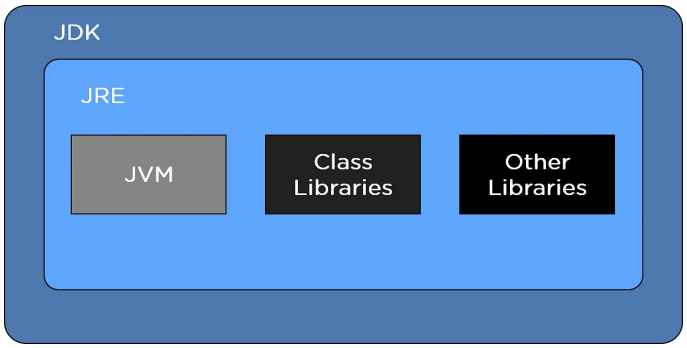
➟ Simple and Easy to learn
➟ Open Source
➟ Platform Independent(Java works on WORA principle)
➟ Secure
➟ High Performance(Just In Time compiler)
➟ Support Multi-Threading
➟ Support Libraries and Frameworks
➟ Robust
➟ Automatic Garbage Collection



Components of Java –

➢ **JDK (Java Development Kit) -** This is a software development kit that includes the tools and libraries needed to develop Java applications. It includes the Java compiler, the Java Virtual Machine (JVM), and other tools and libraries.

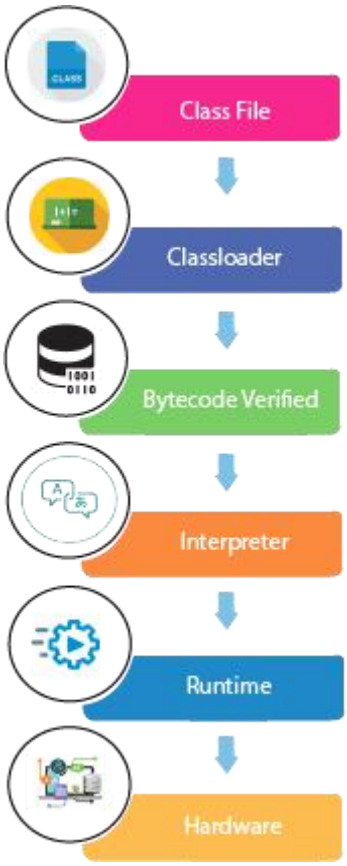
➢ **JRE (Java Runtime Environment) -** This is a runtime environment that provides the libraries and resources needed to run Java applications. It includes the JVM and other components such as class libraries and system classes.



➢ **JVM (Java Virtual Machine) -** This is a virtual machine that is responsible for executing Java bytecode. The JVM interprets Java byte code and executes it on the underlying hardware platform.
JVM provides the functionality of garbage collection, memory management, security, etc.

Note: JVM is an in-built tool that comes along with JDK/JRE and it is responsible for making Java a platform independent language (WORA – Write – Once – Run – Anywhere).

How Java Works -



Step-1 Write a Java code: First, a programmer writes Java code using an Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA. The code is saved in a file with a .java extension.

Step-2 Compile Java code: The Java compiler (JVM) then translates the Java code into bytecode. The bytecode is saved in a file with a .class extension.

Step-3 Load bytecode: When a Java application is run, the JVM loads the bytecode into memory.

Step-4 Execute bytecode: The JVM executes the bytecode, which can include instructions to create objects, call methods, and perform other operations.

Step-5 Garbage collection: Java includes automatic memory management, which means that the JVM automatically manages memory allocation and deallocation. The JVM periodically checks for objects that are no longer in use and frees up memory for other objects.

Step-6 Exception handling: Java includes a robust exception handling mechanism that allows developers to handle errors and exceptions in a structured way. If an error or exception occurs during program execution, the JVM can catch the exception and handle it appropriately.

Step-7 Output results: The Java program can output results to the console, to a file, or to a graphical user interface (GUI).

Compiler -

A compiler is a software tool that translates source code written in a programming language into machine code that can be executed by a computer. Compiler checks the entire program at once and throws all errors at once.
Java is a compiled language.

Interpreter -

An interpreter is a type of software that reads and executes code directly, without first converting it to machine code. Interpreter checks the program line by line.

General Acronyms in Java -

➟ **Single Line comment -** // - Represents a single line comment.
➟ **Multiple Line comment -** /* */ - Represents a multiple line comment.
➟ **End of Statement -** ; - Represents End of a statement.
➟ **Block of Code -** {} - Represents Block of code.

Java Naming Conventions -

Name	Convention
Packages	• The prefix of a unique package name is always written in **all-lowercase**. • Subsequent components of the package name vary according to an organization's Eg: com.nxtgenai.programfundamentals
Classes	• It should be nouns, in mixed case with the first letter of each internal word capitalized. Eg: Capture, LauchApplication
Interfaces	• It should be capitalized like class names.
Methods	• It should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. Eg : bookTickets() , searchUser() etc ..,
Variables	• It should start with lower camel casing Eg: firstName, lastName
Constants(final)	• It should be all uppercase with words separated by underscores Eg: TOTAL_SEAT , MAX_HEIGHT

Main Method -

➟ public static void main(String[] args){} - Code that is inside of the main method will be executed by Java.

Print Method -

➟ System.out.print(); - Prints output in the console window in the same line.
➟ System.out.println(); - Prints output in console window in different lines.

Variables -

Variable is the name of a memory location where we can store different types of data values.

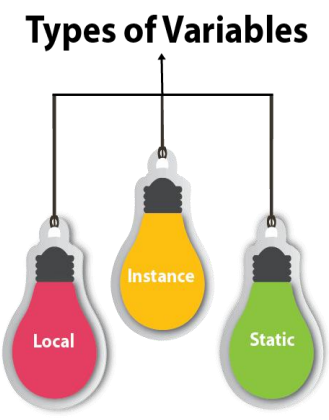
Types of Variables –

➢ **Local Variables -** Variables declared locally inside a method or block of code.

Syntax - dataType variableName = value;

➢ **Static Variables -** Variables declared inside a class but outside method or block of code.
Declared with a "static" keyword.
Static variables can be accessed directly without creating an object of class.

Syntax - static dataType variableName = value;



➢ **Instance Variables -** Variable declared inside a class but outside method or block of code.
Declared without any specific keyword.
Object need to be created to access Instance variables.

Syntax - dataType variableName = value;

Note - Variable names are case-sensitive. "automation" and "AUTOMATION" are different variables).

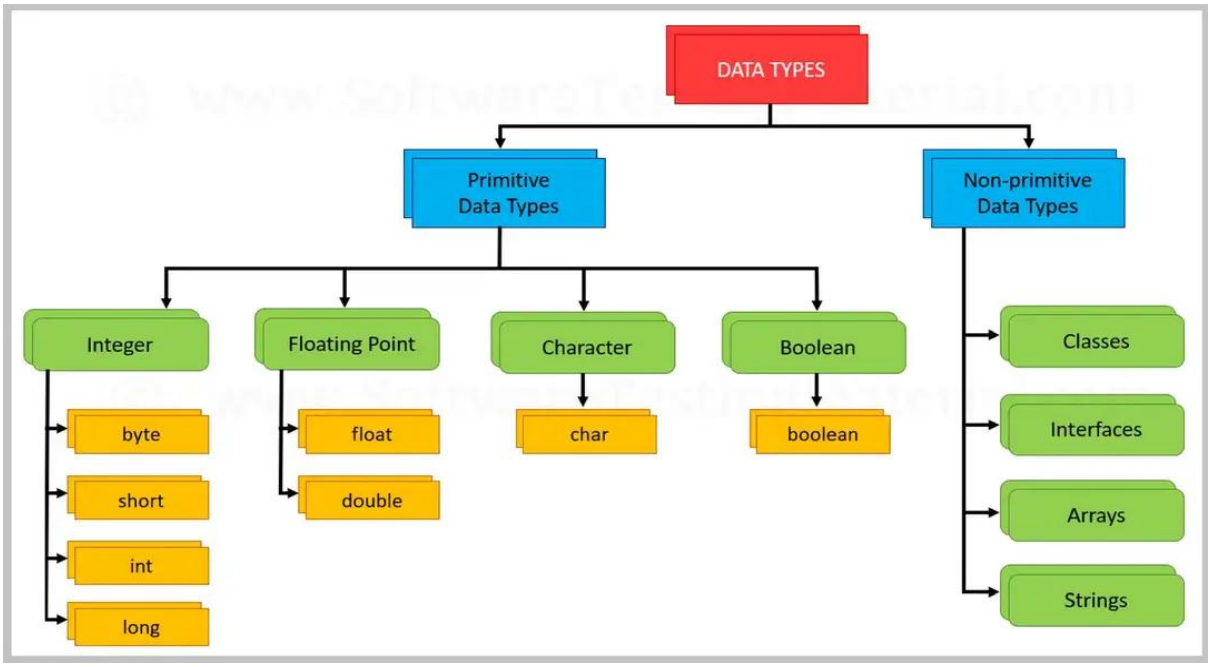
Data Type -

Data type specifies the type of value that a variable is going to store. Java is a statically typed language as data type needs to be specified for every variable.

Types of Data Type -

➢ **Primitive Data Type -** Primitive data type is a basic type of data that is built into a programming language and is not composed of smaller types.

➢ **Non-Primitive Data Type -** Non-primitive data types, also known as reference types, are data types that are not built into a programming language but are created by the programmer. Non-primitive data types are composed of one or



Data Type	Size	Range of values that can be stored	Default value
byte	1 byte	−128 to 127	0
short	2 bytes	−32768 to 32767	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 9223372036854750000	0
float	4 bytes	3.4e−038 to 3.4e+038	0.0f
double	8 bytes	1.7e−308 to 1.7e+038	0.0d
boolean	1 bit	true or false	false
char	2 bytes		\u0000

more primitive data types.

Operators -

Java has several types of operators, which are used to perform various operations on operands (variables, constants, or expressions).

Types of Operators -

➢ **Arithmetic Operators -** Java arithmetic operators are used to perform mathematical operations on numeric values.

➟ **Addition -** Adds two operands (x+y).
➟ **Subtraction -** Subtracts one operand from another (x-y).
➟ **Multiplication -** Multiplies two operands (x*y).
➟ **Division -** Divides one operand by another (x/y).
➟ **Modulus -** Gives the remainder of a division operation (x%y).

➢ **Relational Operators -** Java relational operators are used to compare two values and return a boolean result.

➟ **Greater than -** Returns true if the left operand is greater than the right operand (x>y).
➟ **Less than -** Returns true if the left operand is less than the right operand (x<y).
➟ **Greater than or equals to -** Returns true if the left operand is greater than or equal to the right operand (x>=y).
➟ **Less than or equals to -** Returns true if the left operand is less than or equal to the right operand (x<=y).
➟ **Equals to -** Returns true if the left operand is equal to the right operand (x==y).
➟ **Not equals to -** Returns true if the left operand is not equal to the right operand (x! =y).

➢ **Logical Operators -** Java logical operators are used to combine multiple boolean expressions and evaluate them as a single boolean result.

➟ **Logical AND -** Returns true if both operands are true (x&&y).
➟ **Logical OR -** Returns true if at least one operand is true (x||y).
➟ **Logical NOT -** Reverses the value of a boolean expression (x!y).

Note - Logical Operators are used on boolean expressions and return output as boolean value.

➢ **Increment/Decrement Operators -** Java increment and decrement operators are used to increase or decrease the value of a variable by 1. These operators can be used with integer and floating-point data types.

➟ **Increment operator -** The increment operator '++' adds 1 to the value of a variable (x++ or ++x).
➟ **Decrement operator -** The decrement operator '--' subtracts 1 from the value of a variable (x-- or --x).

Note -
These operators can be used either as a **prefix** or **postfix** operator.
Pre-increment/decrement means the operator first stores value then does increase/decrease.
post-increment/decrement means operator first increase/decrease than store value.

➢ **Assignment Operators -** Assignment operators are used to assign values to variables. They combine the simple assignment operator = with other operators, such as arithmetic operators or bitwise operators.

➟ **Simple assignment (=) -** Assigns the value on the right-hand side to the variable on the left-hand side (x=3).
➟ **Addition assignment (+=) -** Adds the value on the right-hand side to the variable on the left-hand side (x+=3 implies x=x+3).
➟ **Subtraction assignment (-=) -** Subtracts the value on the right-hand side from the variable on the left-hand side (x-=3 implies x=x-3).
➟ **Multiplication assignment (*=) -** Multiplies the variable on the left-hand side by the value on the right-hand side (x*=3 implies x=x*3).
➟ **Division assignment (/=) -** Divides the variable on the left-hand side by the value on the right-hand side (x/=3 implies x=x/3).
➟ **Modulus assignment (%=) -** Calculates the remainder of the variable on the left-hand side divided by the value on the right-hand side (x%=3 implies x=x%3).

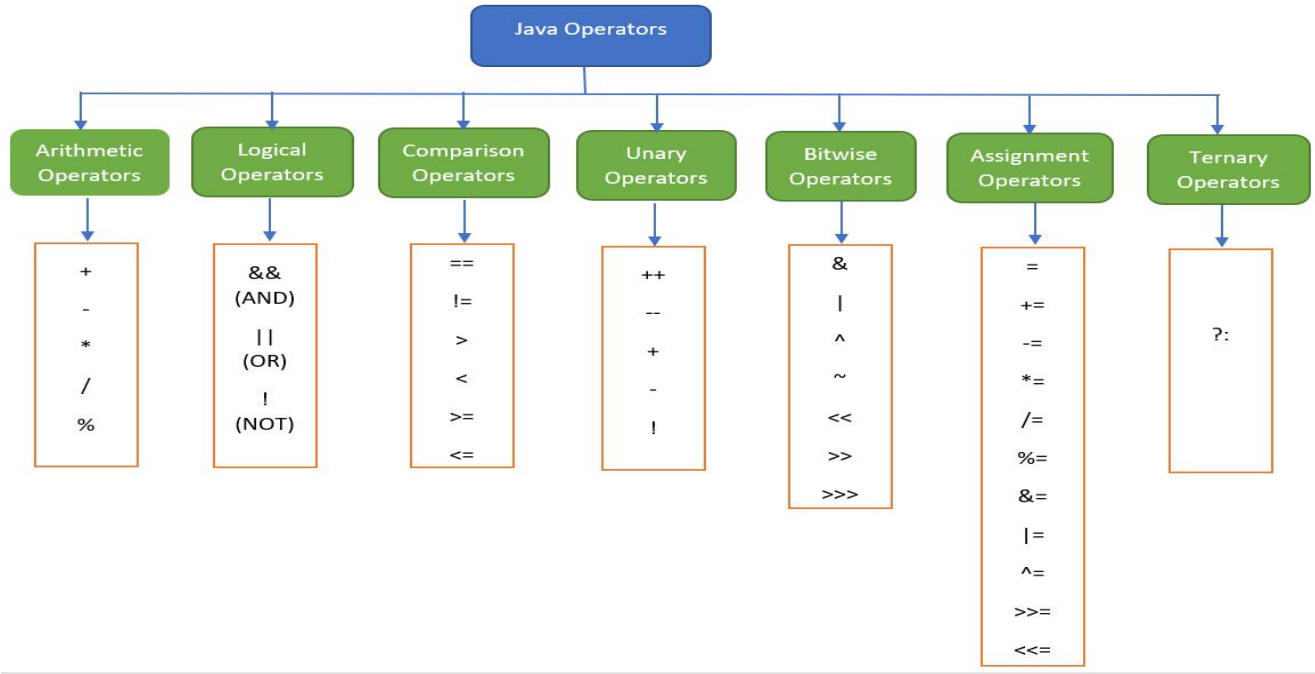
➢ **Ternary Operator -** The ternary operator, also known as the conditional operator, is a shorthand way of writing an if-else statement in Java.
Syntax - variable = (boolean expression)? result1: result2;

Note- If boolean expression is true, output will be result1.If boolean expression is false, output will be result2.

➢ **Unary Operators -** Operators in which operations are performed on 1 single operator are called unary operators.
Example- ++ , -- , =, etc...

➢ **Binary Operators -** Operators in which operations are performed on more than 1 operator are called unary operators.

Example- +, -, *, /, %, >, <, >=, <=, &&, ||, etc...



Control Flow Statements -

In Java, control flow statements are used to control the order in which statements are executed in a program.

There are three main categories of control flow statements in Java:
➟ Conditional Statements
➟ Looping Statements
➟ Jumping Statements

Conditional Statements -

➢ **if Statement -** The if statement is used to execute a block of code only if a specified condition is true.

Syntax-
```
if(condition) {
//Statement
}
```

➢ **if-else Statement -** The if-else statement is used to execute one block of code if a specified condition is true, and another block of code if it is false.

Syntax-
```
if(condition) {
//Statement
}
else{
    //Statement
}
```

➢ **if-else-if Statement -** The if-else-if statement allows you to check for multiple conditions, and execute a block of code if any of those conditions are true.

Syntax-
```
if(condition) {
//Statement
}
Else if(condition){
    //Statement
}
else{
    //Statement
```

```
}
```

➢ **Nested if-else Statement -** The Nested if-else statement allows you to use if else statements within a if else block.

Syntax-

```
if(condition) {
if(condition) {
                //Statement
}
else{
                //Statement
}
}
else{
      //Statement
}
```

➢ **Switch case Statement -** The switch statement allows you to check for multiple conditions and execute a block of code depending on the value of a variable.

Syntax-

```
dataType variable = value;

switch(variable) {
case 1 : Statement-1
break;

case 2 :Statement-2
break;

default : Statement-3
}
```

Note - Switch Case Statements doesn't support String, Boolean, Character, Long data types and Additionally we need to use break; command to halt execution.

Looping Statements -

➢ **while Loop -** while loop in Java enables us to iterate over a number of statements multiple times. It is recommended to use a while loop if we don't know the number of iterations in advance. It is also known as an **entry controlled loop** since the condition is checked at the start of the loop.

Syntax-

```
//Initialisation(dataType variable = value; )
while (condition) {
// Code block to be executed repeatedly while the condition is true
//Increment/Decrement Statement(i++/i-- )
}
```

➢ **do while Loop -** do while loop in Java checks the condition at the end of loop after executing the loop statements. It is recommended to use do while loop if we want to execute the loop at least once even if the condition is false. It is also known as **exit controlled loop** since condition is checked at the end of loop.

Syntax-

```
//Initialisation(dataType variable = value; )
do {
// Code block to be executed repeatedly while the condition is true
//Increment/Decrement Statement(i++/i-- )
}while (condition);
```

➢ **for Loop -** for loop in Java enables us to initialise loop variables, checks the condition and Increment/Decrement variables in a single line of code. It is recommended to use a for loop if the number of iterations are known in advance.

Syntax-

```
//Initialisation(dataType variable = value; )
for(Initialisation; condition; increment/decrement) {
// Code block to be executed repeatedly while the condition is true
}
```

➢ **Enhanced For Loop -** The enhanced for loop, also known as the for-each loop, is a loop that simplifies the process of iterating over arrays and collections in Java. It was introduced in Java 5 and is commonly used when you need to iterate over all the elements in a collection or array.

Syntax-

```
for(dataType variable : collection) {
// Code block to be executed repeatedly while the condition is true
}
```

Jumping Statements -

Jumping Statements in Java are generally used to transfer control to another part of a program depending on condition.

➢ **Break Statement -** If we want to go out of the loop then we use a break statement.
If we use a break statement in a loop then execution will continue with the immediate next statement outside the loop. It will terminate the loop.

➢ **Continue Statement -** If we don't want to execute some statements then we use a continue statement.
If we use a continue statement in a loop then execution will continue from the next iteration. It will not terminate the loop.

Arrays -

Array is a collection of elements of similar data type, we can store multiple values into a single variable. The elements are stored in contiguous memory locations and can be accessed using an index that starts from 0.

Advantage & Disadvantage of Java Arrays -

➢ **Advantages -**
➟ **Code Optimization -** It makes the code optimised, we can retrieve or sort the data easily.
➟ **Random Access -** We can get any data located at any index position.

➢ **Disadvantages -**
➟ **Size Limit (Fixed Size Data) -** If not all the array elements are used, then waste of memory space. If the number of elements to be stored are more than maximum size, the array cannot accommodate those new values.

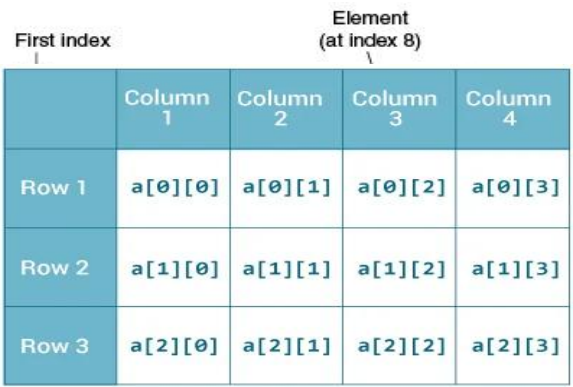
Types of Arrays -

➢ **Single Dimensional Array -** These are the most common type of array in Java, where each element in the array is accessed using a single index value. Data is stored in the form of rows of a table.
The elements in the array are indexed starting from 0 to n-1, where n is the size of the array.

Syntax-

```
datatype[] variable_name = new datatype[size];
datatype[] variable_name = {comma_separated_element_list};
```

➢ **Multi Dimensional Array -** These are arrays that contain multiple rows and columns, allowing for the storage of more complex data. Data is stored in the form of rows and columns of a table.



Syntax-
datatype[][] variable_name = new datatype[row size][column size];
datatype[][] variable_name = {{comma_separated_element_list}{comma_separated_element_list}};

➢ **Object Type Array -** These are arrays that can store different types of data in a single variable.

Methods in Array -

➟ **Length of Array** - array.length() - Returns the length of an array (the number of elements it contains).

➟ **ToString** - Arrays.toString(array) - Converts an array to a string representation.

➟ **Sort -** Arrays.sort(array); - Sorts the elements of an array in ascending order.

➟ **Binary Search -** Arrays.binarySearch(numbers, 20) - Searches for a specific value in a sorted array using the binary search algorithm. Returns the index of the element if found, or a negative value if not found.

➟ **Equals -** Arrays.equals(array1, array2); - Checks if two arrays are equal.

String -

String is a non-primitive data type that is a collection of characters. It is a predefined class in Java that we also use as a data type.

Note- Strings in Java are immutable, which means that once a String is created, it cannot be changed. Any operation that appears to modify a String actually creates a new String object.

Declaration of String -
String variable = "value";
OR
String variable = new String("value");

String Methods -

➟ **Length of String -** s1.length(); **-** Find the number of characters in string.

➟ **Concat -** s1.concat(s2).concat(s3); **-** Joins two string, it can also be done by using '+' operator. But it is suggested to use concat while working with strings.

➟ **Trim -** s2.trim(); **-** Trims white spaces at start and end of String.

➟ **CharAt -** s1.charAt(3); **-** Returns Character based on index.

➟ **Contains -** s1.contains("com"); **-** Returns boolean value on basis of string contains required text or not.

➟ **Equals -** s1.equals(s4); **-** Returns boolean value as a result of comparing 2 strings.

➟ **Equals IgnoreCase -** s1.equalsIgnoreCase(s4); **-** Returns boolean value as output of comparing 2 strings and ignores case of string.

➟ **Replace -** r.replace('r', '#'); **-** Replace one or more characters of strings.

➟ **Sub String -** s1.substring(3,7); **-** Extract substring from main string based on index(Start and End).

➟ **Split -** mail.split("@")[0]; **-** Divides the string into multiple part using delimiter(except- * , $, ^ , ? , . , etc).

➟ **To Upper Case -** c.toUpperCase() **-** Converts entire string to Uppercase.

➟ **To Lower Case -** c.toLowerCase() **-** Converts entire string to Lowercase.

Java User Input -

The Scanner class is used to get user input, and it is found in the java.util package.
To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.

Syntax -

```
Scanner sc = new Scanner(System.in);
String name = sc.nextLine();
int age = sc.nextInt();
```

Access Modifiers -

In Java, access modifiers are used to control the visibility of classes, methods, and variables.

Types of Access Modifiers -

➢ **Public access modifier -** This access modifier allows a class, method, or variable to be accessed from anywhere in the program.
"public" keyword is used.

➢ **Protected access modifier -** This access modifier allows a class, method, or variable to be accessed from within the same package or from a subclass in a different package through inheritance.
"protected" keyword is used.

➢ **Default access modifier -** This access modifier allows a class, method, or variable to be accessed within the same package only.
No keyword required.

➢ **Private access modifier -** This access modifier restricts access to the class, method, or variable to only within the class in which it is defined. Private members cannot be accessed from outside the class.

Access Modifier	Within Class (self)	Within Package (family)	Outside Package by subclass(Inheritance) (relatives)	Outside Package (neighbour)
Public	YES	YES	YES	YES
Protected	YES	YES	YES	NO
Default(No Keyword)	YES	YES	NO	NO
Private	YES	NO	NO	NO

"private" keyword is used.

Wrapper class in Java -

For every built-in data type there is a corresponding class available in Java, that class is called wrapper class.

The wrapper classes are used to convert primitive data types into objects so that they can be manipulated as objects. The eight primitive data types in Java are **byte, short, int, long, float, double, char, and boolean**. Each of these primitive data types has a corresponding wrapper class: **Byte, Short, Integer, Long, Float, Double, Character, and Boolean.**

Converting primitive type into object type is called **autoboxing**.
Converting object type into primitive type is called **unboxing**.

Data conversion methods in wrapper class -

String ---> int => Integer.parseInt()
String ---> float => Float.parseFloat()
String ---> double => Double.parseDouble()
String ---> boolean => Boolean.parseBoolean()
String ---> char => It is not possible

int ---> String => String.valueOf()
float ---> String => String.valueOf()
double ---> String => String.valueOf()
boolean ---> String => String.valueOf()
char ---> String => String.valueOf()

Exception -

In Java, an exception is an event that occurs during the execution of a program, which disrupts the normal flow of the program's instructions. When an exception occurs, the program stops executing.

Types of Exception -

➟ Checked Exception
➟ Unchecked Exception

➢ **Checked Exceptions -** Exceptions that are automatically identified by Java compiler. Java itself provides ways to handle these exceptions[try-catch block and throws keyword].

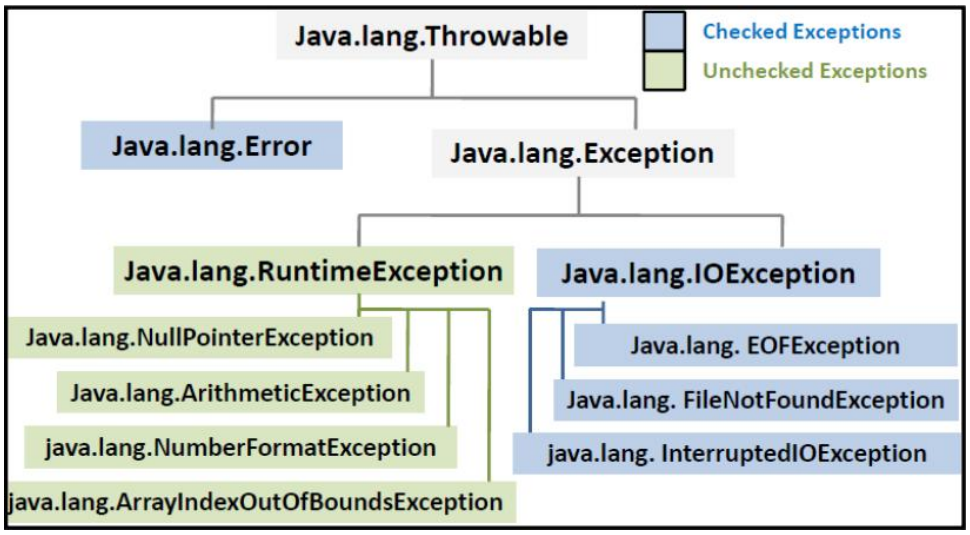
Example - InterruptedExceptions, IOException, FileNotOpen, etc..
➢ **Unchecked Exceptions -** Exceptions that are not automatically identified by Java compiler.

Example - ArithmeticException, NumberFormatException, NullPointerException, ArrayIndexOutOfBound exception, etc…

Exception Handling -

Exception handling is a mechanism in programming that allows developers to handle errors or exceptional situations that occur during the execution of a program.
In Java, exception handling is typically done using **try-catch-finally blocks**. A try block contains the code that may throw an exception, and is followed by one or more catch blocks that handle specific types of exceptions. A finally block is used to execute code that needs to be executed regardless of whether an exception was thrown or not.
Syntax -

```
try {
// Code that may throw an exception
}
catch (ExceptionType1 exception1) {
// Exception handling for ExceptionType1
}
catch (ExceptionType2 exception2) {
// Exception handling for ExceptionType2
}
finally {
// Code that is always executed, whether an exception occurs or not
}
```



Note - If type of exception is known then in catch block instead of Exceptions you can specify the name of exception, it will reduce time of execution but there are few disadvantages like it multiple exceptions cannot be handled in that case.
Note - After Try Catch blocks , users have a choice to use Finally block. Finally block will carry all the statements that need to be executed after checking exception , Finally block will execute no matter if exception is handled or not.

Collections in Java -

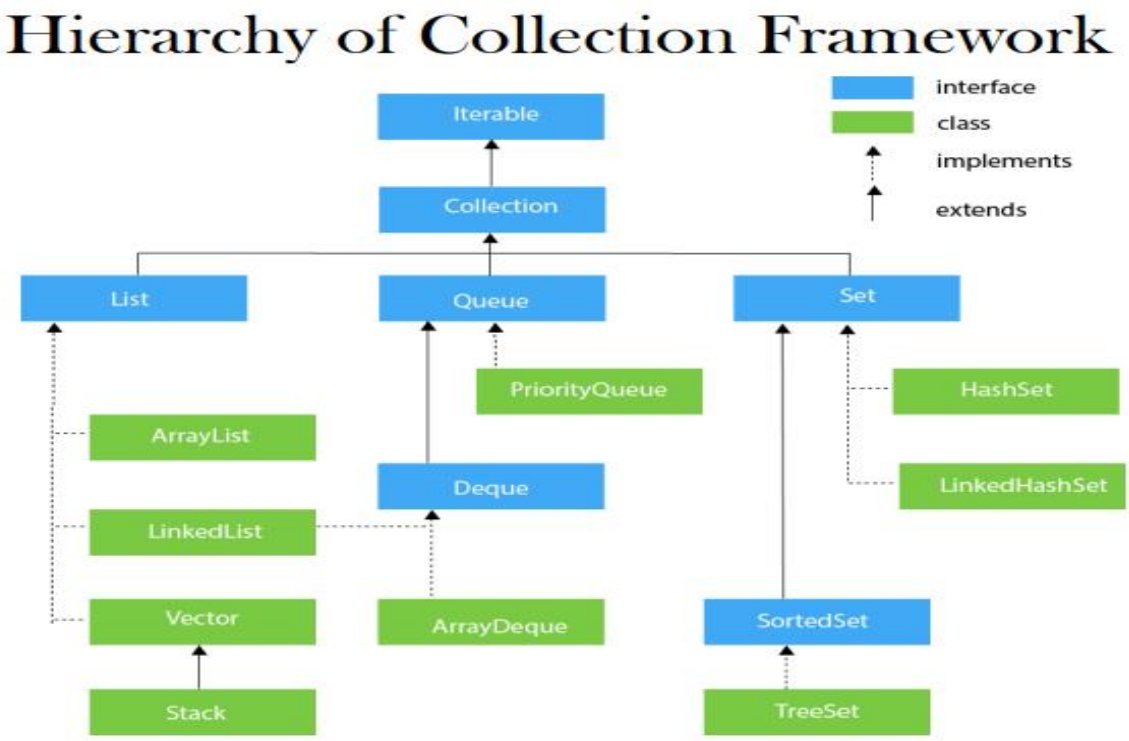
In Java, a collection is an object that groups multiple elements into a single unit. It is used to store, retrieve, and manipulate a group of objects.
Java provides several built-in collection classes in the **java.util** package.

Types of Collections -

➢ **List -** A List is an ordered collection of elements that allows duplicates. It is an interface that extends the Collection interface and provides additional methods to operate on the ordered list of elements

➢ **Set -** A Set is an unordered collection of unique elements, meaning that duplicates are not allowed. It is an interface that extends the Collection interface and provides methods to operate on sets of elements.

➢ **Map -** A Map is a collection that maps keys to values, where each key can map to at most one value. It is an interface that provides methods to operate on the mapping between keys and values.



Array List -

ArrayList is a popular implementation of the List interface in Java, which is a dynamic array that can grow or shrink in size.

Benefits of Array list -

➟ Heterogeneous data is allowed(Different type of data).
➟ Insertion order is preserved(follows index concept).
➟ Duplicate elements are allowed.
➟ Multiple null values are allowed.

Declaration of Array list -

➟ **Homogeneous declaration -** ArrayList mylist = new ArrayList();
➟ **Heterogeneous declaration -** ArrayList <Integer>mylist = new ArrayList<Integer>();

Methods in Array List -

➟ **Adding elements in ArrayList -** mylist.add(element);

➟ **Size of ArrayList -** mylist.size();
➟ **Removing elements from ArrayList -** mylist.remove(Index of element);
➟ **Insert new element in ArrayList -** mylist.add(Index to place element , element);
➟ **Clear elements of ArrayList -** mylist.clear();
➟ **Prints elements of ArrayList -** System.out.println(mylist);
➟ **Get a particular elements of ArrayList -** mylist.get(Index of element);

Hash Set -

A HashSet is a collection class that implements the Set interface. It is used to store a collection of unique elements, meaning that duplicates are not allowed.

Benefits of Hash Set -

➟ Heterogeneous data is allowed(Different type of data).
➟ Insertion order is not preserved(doesn't follows index concept).
➟ Duplicate elements are not allowed.
➟ Multiple null values are not allowed(single null value is allowed).

Declaration of Hash Set -

➟ **Homogeneous declaration -** HashSet myset = new HashSet();
➟ **Heterogeneous declaration -** HashSet<Integer> myset = new HashSet<Integer>();

Methods in Hash Set -

➟ **Adding elements in ArrayList -** myset.add(element);
➟ **Size of ArrayList -** myset.size();
➟ **Removing elements from ArrayList -** myset.remove(Index of element);
➟ **Insert new element in ArrayList -** Not Possible in Hash Set
➟ **Clear elements of ArrayList -** myset.clear();
➟ **Prints elements of ArrayList -** System.out.println(myset);
➟ **Get a particular elements of ArrayList -** Not Possible in Hash Set

Hash Map -

A HashMap is a collection class that implements the Map interface. It is used to store a collection of key-value pairs, where each key must be unique and each value can be duplicated.

Benefits of Hash Map -

➟ Data can be stored in key, value pairs.
➟ Key is Unique but values can be duplicated.
➟ Insertion Order is not preserved(Doesn't follow index concept).

Declaration of Hash Map -

➟ **Homogeneous declaration -** HashMap mymap = new HashMap();
➟ **Heterogeneous declaration. -** HashMap<String,Integer> mymap = new HashMap<String,Integer>();

Methods in Hash Map -

➟ **Adding elements in ArrayList -** mymap.put(key, value);
➟ **Size of ArrayList -** mymap.size();
➟ **Removing elements from ArrayList -** mymap.remove(key of element);
➟ **Insert new element in ArrayList -** Not Possible in Hashmap
➟ **Clear elements of ArrayList -** mymap.clear();
➟ **Prints elements of ArrayList -** System.out.println(myset);
➟ **Get a particular elements of ArrayList -** Not Possible in Hashmap
➟ **Clear value of a key -** mymap.get(key);
➟ **Display all keys as a set -** mymap.keySet();
➟ **Display all values as a set -** Not Possible in Hashmap

OOPS : Object Oriented Programming System

Class -

A class is a blueprint or a template for creating objects. It is a virtual/Logical entity and doesn't occupy memory.
A class defines the properties and behaviours that objects of a particular type will have.
A Java class typically includes fields (variables) to store data and methods (functions) to perform operations on that data.
Syntax - class Student { }

Object -

An object is an instance of a class. It is a real entity and occupies memory to store data and methods.
Objects have state and behaviour. The state of an object is represented by its fields, which hold data. The behaviour of an object is represented by its methods, which define what the object can do.
Syntax - Student obj = new Student();

Object is created using the '**new**' keyword followed by a call to the constructor of the class.
Multiple objects can be created for a single class.

Ways to access variables through objects -

➟ **Using Object Reference variables**
➟ **Using Setter methods**
➟ **Using a constructor**

Methods -

A method is a set of code statements that perform a specific task or action. Methods are used to encapsulate logic and functionality that can be reused throughout a program.

A method can be called through the object reference variable of the class in which method exists.

Types of Methods -

➢ **Built-in Methods -** These are methods that are provided by the Java standard libraries and are built into the language. These methods can be used directly in your code without requiring any additional coding effort.
Example- "println()" , "charAt()" , "sort()" , etc...

➢ **User defined Methods -** These are methods that are created by the programmer to perform specific tasks in their program.
User-defined methods can be declared within a class or outside a class as static methods.

Syntax - AccessModifier ReturnType MethodName(Parameters) { }

Types of User Defined Methods -

➟ Method with 'No Parameter' and 'No Return type'
➟ Method with 'No Parameter' and with 'Return type'
➟ Method with 'Parameter' and 'No Return type'
➟ Method with 'Parameter' and with 'Return type'

Constructor -

A constructor is a special method that is used to initialise objects of a class. The constructor is called automatically when an object is created using the "new" keyword.

In Java, constructors have the same name as the class and do not have a return type, not even VOID.
Constructors can take parameters, which are used to initialise the object's properties. If a class does not have any constructor, then a default constructor is automatically created by the compiler, which does not take any parameters and initialises the object's properties with default values.

Types of Constructors -

➟ Default Constructor
➟ Parameterized Constructor
➟ Copy Constructor

➢ **Default constructor -** It is a constructor that takes no arguments and initialises the object's properties with default values. It is automatically created by the compiler if no constructor is defined in the class.

➢ **Parameterized constructor -** It is a constructor that takes one or more arguments and initialises the object's properties with the values passed as arguments.

➢ **Copy constructor -** It is a constructor that takes an object of the same class as an argument and initialises the new object's properties with the values of the existing object's properties.

Difference between Method and Constructor -

➟ **Purpose -** The primary purpose of a constructor is to initialise the object's state, whereas a method is used to perform some operation or return a value.
➟ **Return type -** Constructors do not have a return type, not even void. They simply create and initialise an object. Methods, on the other hand, must have a return type, which can be void if they don't return anything.
➟ **Name -** Constructors have the same name as the class, whereas methods can have any valid identifier as their name.
➟ **Invocation -** Constructors are invoked automatically when an object of a class is created, whereas methods are called explicitly by the programmer.
➟ **Accessibility -** Constructors can only be invoked from within the class or by another class using the new keyword. Methods can be called from anywhere as long as they are accessible.
➟ **Overloading -** Constructors can be overloaded, which means you can define multiple constructors with different parameters. Methods can also be overloaded, allowing you to define different versions of a method that take different arguments.

Polymorphism -

Polymorphism is a key concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common type, facilitating code reusability, flexibility, and extensibility.

In Java, polymorphism is achieved through method overloading.

Method Overloading -

Method overloading in Java is a feature that allows you to define multiple methods with the same name in a class, but with different parameters. The Java compiler differentiates between these methods based on the number, order, and types of the parameters passed to them.

Rules for method overloading -
➟ Name of the methods should be the same.
➟ Number of Parameters should be different.
➟ Data type of parameters should be different if no. of parameters is same.
➟ If the number and type of parameters are the same , then the order of parameters should be different.

Constructor Overloading -

Constructor overloading in Java is a feature that allows you to define multiple constructors with the same name in a class, but with different parameters. The Java compiler differentiates between these constructors based on the number, order, and types of the parameters passed to them.

Rules for constructor overloading -
➟ Name of the constructor should be the same.
➟ Number of Parameters should be different.
➟ Data type of parameters should be different if no. of parameters is same.
➟ If number and type of parameters are the same , then the order of parameters should be different.

this Keyword -

The "this" keyword in Java is a reference to the current object within a class. It is an implicit reference that is automatically created and maintained by the Java runtime environment for each instance of a class. The "this" keyword can be used inside a class to refer to the current object and its members, such as instance variables and methods.

Encapsulation -

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java —
Step-1 Declare the variables of a class as private.
Step-2 Provide public setter and getter methods to modify and view the variables values.

Note - ***Getter and Setter method can be directly created in Eclipse. Source > Generate Getters and Setters.

Static Keyword -

In Java, the static keyword is used to define a class-level variable or method that can be accessed without creating an instance of the class.

Inheritance -

In Java, inheritance is a mechanism that allows a class to inherit properties and behaviours from a parent class.
The parent class is also known as the superclass, while the class that inherits from the superclass is called the subclass.

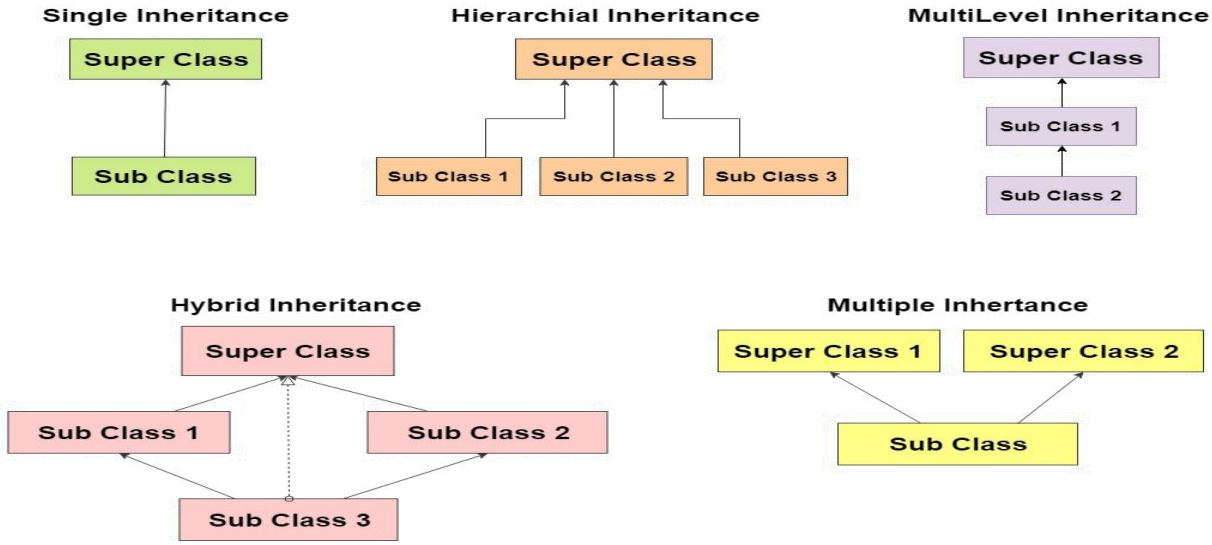
Types of Inheritance -

➢ **Single Inheritance -** In this type of inheritance, a subclass extends only one superclass.

➢ **Multilevel Inheritance -** In this type of inheritance, a subclass extends a superclass, and then another subclass extends that subclass. This creates a hierarchy of classes, with each class inheriting properties and behaviours from the class above it.

➢ **Hierarchical Inheritance -** In this type of inheritance, multiple subclasses extend the same superclass. Each subclass inherits the properties and behaviours of the superclass, but can also add its own properties and behaviours.

➢ **Multiple Inheritance -** In this type of inheritance, a subclass can inherit from multiple superclasses. However, Java does not support multiple inheritance of classes.



Note - **'extends'** keyword is used to inherit superclass.
Note - Multiple inheritance is not supported directly but it can be achieved by Interface because we cannot extend multiple classes into one class because every class has default methods in it and extending multiple classes causes duplication of methods.

Method Overriding -

In Java, method overriding is a mechanism that allows a subclass to provide its own implementation of a method that is already defined in its superclass.

Rules for constructor overloading -

➟ Possible only in multiple classes/inheritance.
➟ Implementation of methods should be same, only body needs to be changed.
➟ The method in the subclass must have the same name, return type, and parameter list as the method in the superclass.

Super Keyword -

In Java, the **super** keyword is a reference variable that is used to refer to the superclass or parent class of the current class. It is often used in the context of inheritance to access or call members (fields or methods) of the superclass from within the subclass.

Super keyword invoked immediate parent call variables and methods and if super keyword will not be used, new overridden value will be printed.

Final Keyword -

In Java, the **final** keyword is used to indicate that a variable, method, or class cannot be changed or overridden.

➟ Variables created with the final keyword cannot be updated because final keyword obsolete user from changing value.
➟ Methods created with the final keyword cannot be overridden from child classes.
➟ Classes created with the final keyword cannot be inherited by child classes.

Data Abstraction -

In Java, data abstraction refers to the process of hiding implementation details from the user and providing only the necessary information to perform a specific task. It is a programming concept that allows programmers to create complex systems without worrying about the underlying complexity.

Ways to Achieve Abstraction -

➟ By using Abstract class
➟ By using Interface concept

➢ Abstract Class -

An abstract class is a class that cannot be instantiated, but can be extended by other classes. It contains abstract methods, which are declared but not implemented in the abstract class. The implementation of these methods is left to the classes that extend the abstract class.

➢ Interface -

➟ An interface is a blueprint of a class.
➟ An interface contains final and static variables(It will take by default, no need to specify explicitly).
➟ An interface contains abstract methods(Also Default methods and Static methods from java8 onwards) and Methods in interface are public.
➟ “interface” keyword is used to define Interface.
➟ A class extends another class, an interface extends another interface but a class implements an interface.
➟ “implements” keyword is used to inherit interface class.
➟ It is not possible to create an object of interface class, so it can be accessed through an object of class implementing interface.

Note - Interface supports functionality of multiple inheritance.

➢ Abstract Method -

An abstract method is a method that contains method definition but not body(Un-implemented methods).
Syntax - void methodName();

Note - Abstract class doesn't provide 100% abstraction on the other hand 100% abstraction can be achieved through Interfaces.

Note - Interface is used when requirements are known but design is not yet available. So we can create empty methods based on requirements and can use later according to design.