1. Introduction to Java

\*History of Java:-

🡪Java: is OOPL (Object Oriented Programming Lang.)is an open source (free to installation and free to updataion) case sensitive.

🡪Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.

🡪*James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

\*Feature of Java:



\* Understanding JVM, JRE, and JDK

JDK : Java Development Kit

JRE : Java Runtime Env.

JVM : Java Virtual Machine

\*Java Program Structure

**1**. Package:-

Definition:  is one type of container .its collection of classes and interface in the first line of the program write package keyword with your folder names like package names

🡪 used same name class like student in a single package but i cant create that class again in same package so i create package another for that class.

**Syntax:**

package mypackage ; // Declares the packag

// Importing a package

import java.util.Scanner; // Single class

import java.util.\*; // All classes in the package

2.class:-

🡪 class an collection of data member(variables) and member function(methods, process) with its behaviors.

**Syntax:-**

class classname

{

data member

member function

}

3 Method:- A **method** is a block of code inside a class that performs a specific task. It is used to define the behavior of objects.



1. Data type,variable and operator:-

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

* Primitive Data type

Non-primitive data type

* 1 Primitive Data type

🡪 A primitive data type is pre-defined by the programming language.

🡪 The size and type of variable values are specified, and it has no additional methods

🡪There are eight primitive data types in Java,



2) Non primitive Datatype The non-primitive data type include class, [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

Non-primitive data types in Java are not predefined. They are created by the programmer.

\*variable declaration and initialization:-

Variable:- Variables are containers for storing data values.

variable declaration:-

Syntax:- dataType variableName;

**Types of Java Variables**

1. Local Variables
2. Instance Variables
3. Static Variables

1 local variable:-A variable defined within a block or method or constructor is called a local variable.

* 1. Instance Variables:-Instance variables are non-static variables and are declared in a class outside of any method, constructor, or bloc
  2. Static Variables :-Static variables are also known as class variables.

\*variable initialization:-

initialization is the process of assigning a value to the Variable.

Every programming language has its own method of initializing the variable. If the value is not assigned to the Variable, then the process is only called a Declaration.

\* Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise:--

Definition: Symbols or keywords that perform operations on variables and values.

Types of Operators:-

1 Arithmetic Operators

2 Relational (Comparison) Operators

3 Logical Operators

4 Bitwise Operators

5 Assignment Operators

6 Unary Operators

1 **Arithmetic Operators**:- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.

🡪They act as basic mathematical operations.

**Operators**: +, -, \*, /, % ,Example: int sum = a + b;

**2 Relational Operators**:- Compare two values and return a boolean result.

**Syntax:** variable1 *relation\_operator* variable2

**Operators:** ==, !=, >, <, >=, <=, **Example**: if (a > b) { ... }

**3 logical operator**:- Logical operators are used to perform logical “AND”, “OR” and “NOT” operations,

**Operators**: &&, ||, !;  **Example**: (a > b) && (c > d)

* **AND Operator**( **&&**) – if( a && b ) [if true execute else don’t]
* **OR Operator** ( **||**) – if( a || b) [if one of them is true to execute else don’t]
* **NOT Operator** (**!**) – !(a<b) [returns false if a is smaller than b]

**4Assignment Operators:-** These operators are used to assign values to a variable. The left side operand of the assignment operator is a variable, and the right side operand of the assignment operator is a value**.**

**Operators**: =, +=, -=, \*=, /=, %=

**5 Unary Operators in Java:-** Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation, etc . **Operators**: +, -, ++, --, !, **Example:** a++;

**6 Bitwise Operators:-** Bitwise operators are used to performing the manipulation of individual bits of a number which can be useful for optimizing performance in certain cases.

🡪They can be used with any integral type (char, short, int).

**\*Type conversion and Type casting:-** type conversion and type casting are mechanisms used to convert data from one type to another.

1 Type Conversion:- Type conversion in Java happens automatically. Data is converted from a smaller type to a larger type.

Type Casting is done during the program design time by the programmer.

2 Type casting:- data type is converted into another data type by a programmer or user while writing a program code of any programming language, the mechanism is known as **type casting**.

**3Control Flow Statements**:-  java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

Java provides three types of control flow statements.

1. Decision Making statements
   * if statements
   * switch statement
2. Loop statements
   * do while loop
   * while loop
   * for loop
   * for-each loop
3. Jump statements
   * break statement
   * continue statement

\*Decision Making statements

* + **if statements:-** Simple if statement:
  + It is the most basic statement among all control flow statements in Java.
  + It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

if(condition) {

statement 1; //executes when condition is true

}

**2) if-else statement:-** The [if-else statement](https://www.javatpoint.com/java-if-else) is an extension to the if-statement, which uses another block of code,

🡪 i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

**Syntax:**

1. **if**(condition) {

statement 1; //executes when condition is true

}

**else**{

statement 2; //executes when condition is false

}

1. **if-else-if ladder:**

🡪The if-else-if statement contains the if-statement followed by multiple else-if statements.

🡪 In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. ---🡪We can also define an else statement at the end of the chain.

if(condition 1) {

statement 1; //executes when condition 1 is true

}

**else** **if**(condition 2) {

statement 2; //executes when condition 2 is true

}

**else** {

statement 2; //executes when all the conditions are false

}

4. **Nested if-statement**

🡪nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. **if**(condition 2) {
4. statement 2; //executes when condition 2 is true
5. }
6. **else**{
7. statement 2; //executes when condition 2 is false
8. }
9. }

**\*Java Switch Statement**

The Java *switch statement* executes one statement from multiple conditions

A control flow statement that allows variable testing against multiple cases.

Simplifies code that requires many conditional checks. Works with data types: int, char, String, and enums.

**Syntax**

switch (expression) {

case value1:

// Code to execute

break;

case value2:

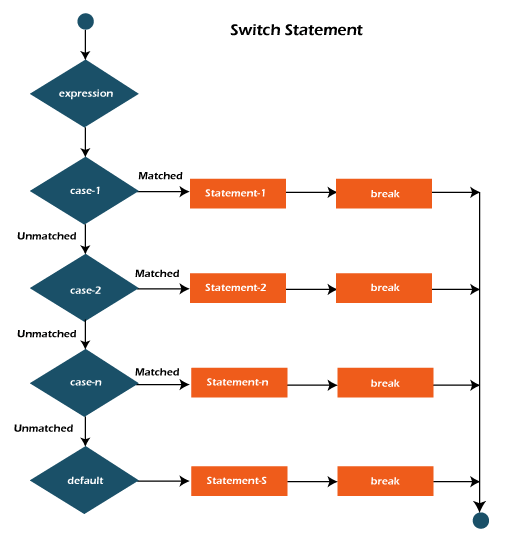
// Code to execute

break;

default:

// Code to execute if no match

}

**Flowchart of Switch Statement** 

**\*** **Java for Loop:-** For loops in Java are a fundamental control structure used to repeat a block of code a specific number of times or iterate through a sequence of values.

🡪 **Syntax:** for(initialization; condition; increment/decrement){

//statement or code to be executed

}

**Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

**Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

**Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.

**Statement:** The statement of the loop is executed each time until the second condition is false.

\* **The Java while loop:-** while loop is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops**.**

**\* Syntax:**

while (condition){

//code to be executed

Increment / decrement statement    }

1. **Condition:** It is an expression which is tested. If the condition is true, the loop body is executed. When the condition becomes false, we exit the while loop.
2. **Update Expression:** Every time the loop body is executed, this expression increments or decrements loop variable.

**\* Java do-while Loop**

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true Java do-while loop is called an **exit control loop.**

**Syntax:**

**do**{

//code to be executed / loop body

//update statement

}**while** (condition);

**1 Condition**: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression.

**2. Update expression**: Every time the loop body is executed, the this expression increments or decrements loop variable.

**\* Break and Continue Keywords:-**

The `break` statement in Java is a powerful tool for controlling the flow of your program. It is used to exit a loop or switch statement prematurely.

The Java *break* statement is used to break loop or [switch](https://www.javatpoint.com/java-switch) statement.

🡪 It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

**Java continue Statement**

🡪 The continue statement skips the current iteration of a loo

🡪After the continue statement, the program moves to the end of the loop. And, test expression is evaluated (update statement is evaluated in case of the for loop)

* 1. **Classes and Object**
* **1) class** : class is an collection of data member(variables) and member function(methods, process) with its behaviors
* An instance of a class that uses the properties and methods defined in the class.
* **syntax**

class classname

{

data member

member function

}

2) object : object is a instancs of an class .

🡪when you create class variables also called..

🡪:its uses new keyword and class constructor to create object

🡪:access whole properties of an class except private

* **syntax**

classname objectname = new constructor();

**\*constructor and overloading**

**🡪**Constructor is simple type of method whose name is same as class name.

🡪The constructor is called when an object of a class is created.

🡪There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

🡪It is a special type of method which is used to initialize the object

**Syntax:-** class ClassName {

ClassName() {

// Constructor body

}

}

**\* Types of Java Constructors**

* 🡪There are two types of constructors in Java:

1. Default Constructor (no-arg constructor)
2. Parameterized Constructor

* **1) default :** no any argument in constructor
* **2) parameterized:** may have one or more argument in constructor

**\*Constructor Overloading**

**🡪** The practice of defining multiple constructors in a class with different parameter lists.

🡪Allows objects to be initialized in various way

**\*Object Creation, Accessing Members of the Class**

Object is called instance of class.

**Syntax of Object Creation**

ClassName obj = new ClassName();

**\*This keyword :-**

🡪A reference variable that refers to the current object.

🡪Used within a class to differentiate between class members and parameters.

\***Uses of this Keyword:-**

🡪To refer to the current object.

🡪To resolve naming conflicts between class fields and method/constructor parameters.

🡪To call another constructor in the same class (constructor chaining).

🡪To pass the current object as a parameter.

To return the current object.

**5 Methods in Java:-**

🡪a method is a way to perform some task. the method in Java is a collection of instructions that performs a specific task.

🡪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’

**\*method parameter**

**🡪**parameters are specified after the method name, inside the parentheses.

🡪You can add as many parameters as you want, just separate them with a comma.

**Types:**

* **Formal Parameters:** Defined in the method signature.
* **Actual Parameters (Arguments):** Values passed to the method when it is called.

**Syntax**

returnType methodName(parameterType parameterName) {

// method body

}

**\*Return Types:-** The return type specifies the type of value the method returns. If a method does not return a value, the return type is void.

**Syntax :-**

returnType methodName(parameters) {

// method body

return value; // for non-void return types

}

**\*Method Overloading**

**🡪**Method overloading in Java is a feature that allows a class to have more than one method with the same name but different parameters (argument lists).

Method overloading in Java is also known as[Compile-time Polymorphism](https://www.geeksforgeeks.org/compile-time-polymorphism-in-java).

\***Static Methods and Variables in java**

**🡪**static methods and static variables are part of the class rather than any specific instance of the class.

🡪They belong to the class and are shared among all instances of the class.

🡪The **static keyword** in java  is used for memory management mainly. We can apply static keyword with variable, methods, blocks .

🡪The static variable can be used to refer to the common property of all objects

**5 Object-Oriented Programming (OOPs) Concepts:-**

**🡪**object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects.

**.1 Encapsulation:-** Encapsulation is the process of wrapping data (variables) and methods (functions) together as a single unit and restricting direct access to some of the object's components.

.**2 Inheritance:-** Inheritance allows one class (child or subclass) to inherit the properties and methods of another class (parent or superclass). This promotes code reuse and hierarchical classification.

:there are mainly 5 types

1) single : only one parent having only one child

2) multilevel : single inheritance having one another child

3) hierarchical : one parent having 2 or more child

4) multiple : java does not support directly

5) hybrid: java does not support directly

**3 Polymorphism**

🡪Polymorphism means "many forms." In Java, it allows methods to perform different tasks based on the object calling them. There are two types:

1. **Compile-time Polymorphism (Method Overloading)**: Methods with the same name but different parameter lists.
2. **Runtime Polymorphism (Method Overriding)**: A subclass provides its specific implementation for a method defined in its superclass.

**4. Abstraction**

Abstraction is the process of hiding implementation details and showing only the essential features of an object. It is achieved in Java using:

* **Abstract Classes**: Declared with the abstract keyword.
* **Interfaces**: Use interface keyword to define a contract for implementing classes.

**\*Inheritance: Single, Multilevel, Hierarchical**

🡪Inheritance is one of the fundamental principles of Object-Oriented Programming (OOP).

🡪It allows a class (subclass or child class) to inherit properties and behaviors (methods) from another class (superclass or parent class).

**Single Inheritance:** One subclass inherits from one superclass.

**Multilevel Inheritance:** A subclass inherits from a superclass, and another subclass inherits from this subclass.

**Hierarchical Inheritance:** Multiple subclasses inherit from a single superclass.

**\*Method Overriding and Dynamic Method :-**

**🡪**Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass

**Dynamic Method Dispatch**

**Dynamic Method Dispatch**, also known as **runtime polymorphism**, is a mechanism in Java where the method that gets called is determined at runtime based on the object that the reference points to.

.**7Constructors and Destructors**

\*Constructor Types :-

**1. Default Constructor**

A **default constructor** is provided by the Java compiler if no constructors are explicitly defined in the class. It initializes object attributes with default values.

2 **Parameterized Constructor**

A **parameterized constructor** accepts arguments to initialize object attributes with specific values at the time of object creation.

**\*Copy Constructor (Emulated in Java)**

**🡪A copy constructor is a special type of constructor that creates a new object by copying the attributes of an existing object.**

**\*Constructor Overloading:-**

Constructor overloading occurs when a class has more than one constructor, each with different parameter lists.

🡪 This allows objects of the class to be initialized in different ways depending on the constructor used.

**\*Object Life Cycle and Garbage Collection**

**🡪**Java is an object, and objects go through many stages in their lifetime.

🡪In order to ensure proper resource management and program functioning, Java developers need to be aware of the object life cycle.

**Creation:** Object is instantiated.

**Usage:** Object methods and properties are accessed.

**Dormancy:** Object becomes unused (eligible for garbage collection).

**Destruction:** Object is removed from memory.

**\*Garbage Collection:-**

**🡪**Garbage collection in Java is the process by which Java programs perform automatic memory management.

🡪 Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short

**8 Arrays and Strings**

One-Dimensional and Multidimensional Arrays

* **1 Single Dimensional Array**
* These are the most common type of arrays, where elements are stored in a linear order
* At a time only one loop will be use.
* SYNTAX:- int[]= arr= new int[5];
* **2 Multidimensional Arrays**
* A multidimensional array is an array of arrays.
* Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.
* **DECLARATION SYNTAX**
* dataType[][] arrayName;

**\*String Handling in Java: String Class, StringBuffer, StringBuilder**

**String :** immutable : we can not changed

:its provide so many methods to use it for your program

:set of sequence of characters .

🡪The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable.

🡪There are many differences between StringBuffer and StringBuilder

\* **String Buffer**

* **🡪** StringBuffer is similar to StringBuilder in terms of mutability and efficient string manipulation.
* **🡪** However, it is thread-safe, meaning multiple threads can safely use a StringBuffer without the risk of data corruption**.**

**\* String Builder**

🡪 a mutable alternative to String. It allows for efficient string manipulation, such as appending, inserting, or modifying characters within the string, without creating new objects**.**

**\*** **Array of Objects in java**

array of objects is an array that stores references to objects, instead of primitive types

🡪. It allows you to group a collection of objects of the same class.

\* **Syntax**: ClassName[] arrayName = new ClassName[size];

**\* String Methods:-**

String class provides numerous methods for manipulating and handling strings.

🡪Below is a list of commonly used methods categorized for better understanding:

**Java String length()**

🡪The length() method returns the number of characters in a [string](https://www.programiz.com/java-programming/string).

**length() Syntax**

string.length()

**Java String charAt()**

The charAt() method is a fundamental tool in Java for accessing individual characters within a string.

the charAt() method, exploring its syntax, functionality, and common use cases.

**Syntax**

1. public char charAt(int index)

# **substring() Method**

🡪The substring() method returns a substring from the string.

🡪If the *end* argument is not specified then the substring will end at the end of the string.

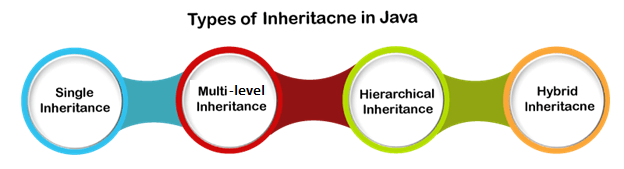
**Syntax**

public String substring(int start, int end)

1. **Inheritance and Polymorphism**

**🡪**inheritance allows a class to inherit properties and behaviors (fields and methods) from another class.

🡪This promotes code reusability and establishes a hierarchical relationship between classes**.**

**Single Inheritance**

🡪 In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class.

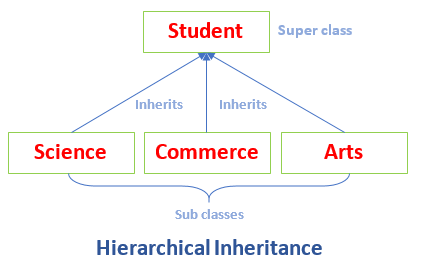
**Multi-level Inheritance**

In **multi-level inheritance**, a class is derived from a class which is also derived from another class is called multi-level inheritance.

🡪 In simple words, we can say that a class that has more than one parent class is called multi-level inheritance.

**Hierarchical Inheritance**

If a number of classes are derived from a single base class, it is called **hierarchical inheritance**.



**Hybrid Inheritance**

--Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance

**Benefits of Inheritance**

**Code Reusability:** Inherited members from a superclass can be reused in subclasses, reducing redundant code and promoting a modular approach to software development.

**Hierarchical Organization:** Inheritance facilitates the creation of well-structured class hierarchies, improving code readability and maintainability.

**Polymorphism:** Subclasses can override superclass methods, allowing for polymorphic behavior, where methods can behave differently based on the object type at runtime.

**Easier Maintenance:** Changes made to a superclass automatically propagate to its subclasses, ensuring consistency and simplifying maintenance efforts.

**\*Method Overriding**

🡪 subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

-🡪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.

\***Dynamic Binding (Run-Time Polymorphism**)

🡪Dynamic binding, also known as **run-time polymorphism**, is a process where the method to be executed is determined at runtime, not at compile time. This is primarily achieved through **method overriding** in Java.

**\*characteristics**

1. **Inheritance**:
   * Run-time polymorphism relies on inheritance, where a subclass inherits methods from its parent class.
2. **Method Overriding**:
   * The subclass provides its own implementation for a method defined in the parent class.
3. **Reference Type vs. Object Type**:
   * The method that gets invoked depends on the actual type of the object, not the type of the reference.
4. **Happens at Runtime**:
   * The JVM decides which method to invoke based on the object type during runtime.

**\*Super Keyword:-**

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

-🡪Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable**.**

**\*Method Hiding in java**

**🡪**Method Hiding in Java occurs when a subclass defines a static method with the same name and signature as a static method in the parent class. In this case, the method in the child class hides the method in the parent class, rather than overriding it.

1. **Interfaces and Abstract Classes**

**\*Abstract Classes and Methods:-**

🡪abstract class in Java is a class that is declared with the abstract keyword. It serves as a blueprint for other classes and cannot be instantiated directly.

🡪Abstract classes are primarily used to define a common interface and share code among subclasses while enforcing certain behaviors.

\* **Abstract Methods in Java**

An **abstract method** in Java is a method that is declared but does not have an implementation.

🡪It serves as a blueprint, forcing subclasses to provide their own specific implementation for that method.

**\* Interfaces: Multiple Inheritance in Java**

**Interfaces in Java**

🡪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 provide a way to achieve **abstraction** and **multiple inheritance** in Java.

**\* Multiple Inheritance in Java**

**Multiple inheritance** is a feature in some programming languages where a class can inherit characteristics and behaviors (methods) from more than one parent class.

**Implementing Multiple Interfaces** :-

Multiple inheritances can be achieved through the use of interfaces. Interfaces are similar to classes in that they define a set of methods that can be implemented by classes.

1. **Packages and Access Modifiers**

🡪 package is a namespace that organizes a set of related classes and interfaces.

🡪 Conceptually, you can think of a package as a folder in a file directory, where you can group related files (classes and interfaces) together.

🡪Packages help avoid name conflicts and can control access with the use of access modifiers**.**

**1 Built-in Packages**

Built-in packages are provided by the Java Development Kit (JDK) and contain a vast number of classes and interfaces that are commonly used in Java programming. These packages are organized into a hierarchical structure and can be used without having to create them manually.

**2 User-defined Packages**

**User-defined packages** are created by developers to organize their own classes and interfaces. This is especially useful in larger applications to avoid naming conflicts and to group related classes together.

**\*Access Modifiers**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**\* Importing Packages and Classpath**

\*  the import keyword used to import built-in and user-defined packages. When a package has imported, we can refer to all the classes of that package using their name directly.

🡪The import statement must be after the package statement, and before any other statement.

\* **Classpath:-**

The **classpath** is a parameter that tells the Java Virtual Machine (JVM) and Java compiler where to find user-defined classes and packages in Java programs.

🡪It includes directories, JAR files, and ZIP files that contain the classes and packages that your application needs.

**12. Exception Handling:-**

**Types of Exceptions:**

**Checked and Unchecked**

**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.

**2 Unchecked Exceptions**

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.

**\*try, catch, finally, throw,**

exception handling is a powerful mechanism that allows developers to manage errors and other exceptional events that occur during program execution.

**1 try**

The try block is used to wrap the code that may throw an exception. If an exception occurs, the normal flow of the program is interrupted, and control is transferred to the corresponding catch block.

2 **catch**

The catch block is used to handle the exception that may be thrown by the try block. You can have multiple catch blocks to handle different types of exceptions.

**3. finally**

The finally block is used to execute code regardless of whether an exception occurred or was caught. It's typically used for cleanup activities, such as closing files or releasing resources.

**4. throw**

The throw statement is used to explicitly throw an exception from a method or a block of code. You can throw both checked and unchecked exceptions.

**5. throws**The throws keyword is used in a method declaration to indicate that a method may throw one or more exceptions. It allows the caller of the method to handle these exceptions.

**\*Custom Exception Classes**

**🡪**custom exception classes to handle specific error conditions that are relevant to your application.

🡪 Custom exceptions help you define your own exception types, making your code more readable and easier to maintain

**13Multithreading:-**

**🡪**A Thread is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

In order to perform complicated tasks in the background, we used the Thread concept in Java.

🡪 All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.

**\*Creating Threads by Extending Thread Class or Implementing Runnable Interface**

the Thread class, you create a new class that inherits from Thread. You need to override the run() method to define the code that will be executed in the new thread.

1. Create a new class that extends Thread.
2. Override the run() method.
3. Create an instance of your class and call the start() method to begin execution**.**

**\* Implementing Runnable Interface**

Java runnable is an interface used to execute code on a concurrent thread. It is an interface which is implemented by any class if we want that the instances of that class should be executed by a thread.

🡪The runnable interface has an undefined method run() with void as return type, and it takes in no arguments.

**\* Thread Life Cycle**

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

**1 New (Created)**

* A thread is in the new state when it is created but not yet started.
* This happens when a Thread object is instantiated using the Thread class or by implementing the Runnable interface.
* The thread remains in this state until the start() method is invoked.

**2 Runnable (Ready to Run)**

* When the start() method is called, the thread moves to the **runnable state**.
* The thread is now ready to run and is waiting for CPU time to execute.
* The thread scheduler decides when the thread will actually run.

**3 Running**

* The thread is in the running state when the CPU starts executing its run() method.
* A thread can enter this state only if it has been selected by the thread scheduler from the runnable state.

**Blocked/Waiting/Timed Waiting**

* A thread enters these states when it cannot proceed further for some reason:
  + **Blocked**: Waiting for a resource to be released (e.g., I/O or synchronization locks).
  + **Waiting**: Waiting indefinitely for another thread to signal or notify it.
  + **Timed Waiting**: Waiting for another thread to signal or notify it, but only for a specified time.

**5. Terminated (Dead)**

* A thread enters the terminated state when it finishes its execution or is explicitly stopped.
* Once a thread is terminated, it cannot be restarted.

**\* Synchronization in Java**

**Synchronization** in Java is a critical concept in concurrent programming that ensures multiple threads can interact with shared resources safely.

**🡪** In a nutshell, synchronization prevents race conditions, where the outcome of operations depends on the timing of thread execution.

**\*Inter-thread communication**

**Inter-thread communication in Java** allows threads to communicate with each other and coordinate their actions. This is especially useful when multiple threads are working on shared resources and need to synchronize their activities.

Java provides built-in methods for inter-thread communication in the Object class:

* wait()
* notify()
* notifyAll()🡪These methods must be used within synchronized blocks or methods to ensure thread safety.
* **14. File Handling**

🡪The File class can be used by creating an object of the class and then specifying the name of the file.

🡪 **File Handling** in Java is the process of performing various operations (like creating, reading, writing, and deleting) on files

**(java.io package):-**

🡪the **java.io package** in Java provides classes and interfaces for handling various types of input and output operations, including reading and writing data to files, handling standard I/O streams, and working with objects, arrays, and characters. Here's an overview of the key components of the java.io package

**\*FileReader and FileWriter Classes**

The FileReader and FileWriter classes in Java, part of the java.io package, are used for reading and writing character data to and from files.

**\* File Writer Class:-**

🡪Java FileWriter class is used to write character-oriented data to a file

🡪It is character-oriented class which is used for file handling in java .

\* **FileReader** **class**

The **FileReader** class in Java, part of the java.io package, is designed for reading character files.

🡪 It reads character streams, making it ideal for handling text files. It is a convenience class for reading text data and inherits from the InputStreamReader class.

**\* BufferedReader**

The **BufferedReader** class in Java, part of the java.io package, is a character-based class that reads text from an inputstream. It buffers characters to provide efficient reading, especially for large files or data streams.

🡪 It is often used for reading lines of text efficiently.

**\* BufferedWriter**

**🡪** BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast

🡪. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).

**\* Serialization and Deserialization**

**Serialization:-**

Serialization in Java is the process of converting an object's state into a byte stream so that it can be saved to a file, transferred over a network, or stored in a database. The reverse process, converting the byte stream back into an object, is called deserialization.

**Deserialization:-**

**Deserialization** in Java is the process of reconstructing an object from a serialized byte stream. It allows us to restore the state of an object that was previously serialized and saved to a file, transferred over a network, or stored in another medium.

**15 . Collections Framework**

**Introduction to Collections Framework**

🡪The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

🡪Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion**.**

🡪Java Collection means a single unit of objects.

**List, Set, Map, and Queue Interfaces**

**1. List Interface**

The List interface represents an ordered collection (also known as a sequence) that allows duplicate elements. It provides methods to manipulate the size of the list, access elements by their index, and perform various operations.

**2 Set Interface**

The Set interface represents a collection that does not allow duplicate elements. It models the mathematical set abstraction and is used to store unique elements.

**3 Map Interface**

The Map interface represents a collection of key-value pairs, where each key is unique, and each key maps to exactly one value. It is not a part of the Collection hierarchy.

**4. Queue Interface**

The Queue interface represents a collection designed for holding elements prior to processing. It is used primarily to implement data structures such as queues.

\***ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap in java**

**List :**-

List is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

**ArrayList:-**

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types.

🡪 The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed

**LinkedList:-**

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements.

🡪It can store the duplicate elements. It maintains the insertion order and is not synchronized.

**HashSet:-**

HashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

**HashMap:-**

HashMap in Java is a key-value data structure offering efficient data access via keys using hashing.

🡪Hashing converts large strings or other objects into smaller, consistent values for quick indexing and searching

🡪. HashMap implements the Map interface and is used for managing large datasets efficiently

**TreeSet:-**

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements.

**TreeMap**

**Java TreeMap** is a red-black tree-based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The java.util package contains the Java TreeMap class, which is a component of the Java Collections Framework.

**Iterators and ListIterators**

**Iterators**

The Java Iterator is also known as the **universal cursor** of Java as it is appropriate for all the classes of the Collection framework. The Java Iterator also helps in the operations like READ and REMOVE. When we compare the Java Iterator interface with the enumeration iterator interface

**ListIterators**

ListIterator is an interface in Collection API. It extends Iterator interface. To support Forward and Backward Direction iteration and CRUD operations, it has the following methods. We can use this Iterator for all List implemented classes like ArrayList

**16. Java Input/Output**

Java, streams are the sequence of data that are read from the source and written to the destination.

An **input stream** is used to read data from the source. And, an **output stream** is used to write data to the destination.

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

**OutputStream**

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

-🡪OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

**\* Handling File I/O Operations**

Handling File I/O (Input/Output) operations in Java involves using the classes in the java.io and java.nio packages.

🡪These provide a framework for reading from and writing to files, handling text or binary data.

🡪 File handling in Java implies reading from and writing data to a file. The File class from the java.io package, allows us to work with different formats of files. In order to use the File class, you need to create an object of the class and specify the filename or directory name