**## What is a java ?. 🡪**Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language. Java was developed by *Sun Microsystems*  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 Oak name to Java. **advantage of java programming** --Object-Oriented Programming language – simple syntax for easier learning. –The standard of enterprise computing – Extensive talent pool -- Secure language. –distributed language –Write once run anywhere --Automatic memory management --community support --Supports multithreading .

**Features of 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*. --Simple --Object-Oriented --Portable --Platform independent --Secured --Robust --Architecture neutral --Interpreted --High Performance -- Multithreaded --Distributed --Dynamic **Distributed :** Java has features like Remote Method Invocation (RMI) and Enterprise JavaBeans (EJB) that enable distributed computing, where applications can communicate and share resources over a network. **Dynamic** : Java is dynamic in nature, as it can dynamically link new class libraries, methods, and objects during runtime. It also supports dynamic compilation, which helps in adapting to changing environments. **Simple :**  Java is very easy to learn and its syntax is simple clean and easy to understand to c++ it make familiar to developed.

**## Wha is OOP ?**  🡪In OOP, the major emphasis is on data rather than procedure (function). It ties data more closely to the function that operate on it, and protects it from accidental modification from outside function. OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects. The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects. C++, Java, Dot Net, Python etc are the example of Object oriented programming (OOP) language. **Characteristic of OOP : --**Emphasis is on data rather than procedure (function). --Programs are divided into objects --Functions that operate on the data of an object are ties together in the data structure. --Data is hidden and cannot be accessed by external function. --Objects may communicate with each other through function. --New data and functions can be easily added whenever necessary. --Follows bottom up approach in program design.

**Features of OOP:**  **Class:** A blueprint for creating objects, defining a set of properties (fields) and behaviors (methods) that the objects created from the class will have.  **Object:** An instance of a class. Each object has its own state and behavior defined by the class.  **Overloading:** Allows a class to have more than one method with the same name but different parameter lists. It is a form of compile-time polymorphism.  **Overriding:** Allows a subclass to provide a specific implementation of a method that is already provided by its superclass. It is a form of runtime polymorphism .  **Interface:** A reference type in Java, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces are used to achieve abstraction and multiple inheritance.  **Abstract Class:** A class that cannot be instantiated on its own and is intended to be subclassed. It can contain both abstract methods (without an implementation) and concrete methods (with an implementation)

**## Procedure Oriented Programming (POP);**  🡪In the procedure oriented approach, large programs are divided into smaller programs known as functions. In POP, a program is written as a sequence of procedures or function. In POP, each procedure (function) contains a series of instructions for performing a specific task. During the program execution each procedure (function) can be called by the other procedures. To call a procedure (function), we have to write function name only. Examples of procedural oriented programming language are COBOL, FORTRAN, PASCAL, C programming language etc.

**Characteristic of POP: --**Large programs are divided into smaller programs known as functions. --Most of the functions share global data. --Data move openly around the system from function to function. --Functions change the value of data at any time from any place. (Functions transform data from one form to another.) --It uses top-down approach in program design. **## Advantage of OOP:**  --The programs are modularized based on the principles of classes and objects. –linking code & objects allows related objects to share common code. This reduces code duplication and code reusability. –Creation and implementation of OOP code is easy and reduces software development time. –The concept of data abstraction separates object specification and object implementation. –Easier to develop complex software because complexity can be minimized through inheritance

|  |  |
| --- | --- |
| **OOP** | **POP** |
| Program is divided into objects | Program is divided into functions |
| Bottom – up approach | Top down approach |
| Inheritance property is used | Inheritance is not allowed |
| It uses access specifier | It doesn’t use access specifier |
| Encapsulation is used to hide the data | No data hiding |
| The existing code can be reused | No code reusability |

**## What is JVM ? 🡪**The Java Virtual Machine (JVM) is a software-based engine that runs Java applications by executing bytecode, which is a platform-independent code generated from Java source code. When you write a Java program, the Java compiler converts it into bytecode (.class files), and the JVM is responsible for interpreting or compiling this bytecode into machine code that can be executed by the host operating system.

**## Java Variables:** A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location. There are three types of variables in java: **--local variable** : It declared inside of methods, constructors or blocks. Access modifier cannot be used for local variables. Local variables are visible only within the methods. Variable can be accessed when the method is called. Eg. Class { method { declare variable } } **--instance variable** : It declare inside of a class but outside of methods. This variables can be access by any methods. By creating object for a class we can access this variable from main methods. E.g. : class { declare variables methods { // can use var } } **--class / static variable:**  Variables declared inside of class but outside of methods with static keywords. Static variables are stored in the static memory. Static variables can be accessed by static methods or constructors. Syntax : class { declare static variables constructor { // can use var } static method { //can use var } }

**## Operator : 🡪** Operators are special symbols that perform specific operations on one ,two or three operands, and then return a result. Operators with one operand are called unary operators. Operators with two operands are called binary operators. Operators with three operands are called ternary operator . **Operators available in java: Assignment Operator :** The = is called the assignment operator. The = operator is used to assign the value present to its right to the operands present to its left. Eg.: int a=10 , b; b=a; **Arithmetic Operator:**  The basics arithmetic operator are :

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulo |

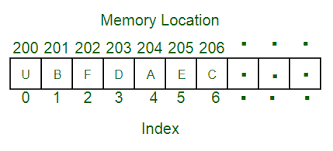
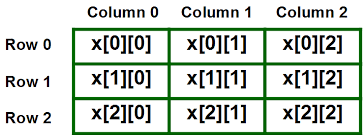
**String concatenation operator :** The + sign can also be used to concatenate two strings. E.g.: String a =” hello “ , b=” world “; system.out.println (a+b) will result as helloworld **Increment/ decrement operator:**  The ++ is the increment operator. The – is the decrement operator. There are two types based on the position of the operator with the operand. Postfix : ++ or -- to the right of the operands Prefix : ++ or -- to the left of the operands.  **Conditional operator :**  The conditional operator is the ternary operator. The syntax of conditional operator is x= (Boolean expression ) ? true : false ;  **Logical Operator:**  Bitwise operator : &, | , ^ Shorthand operators : && logical AND || logical OR

**## Method : 🡪**A **method** is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as **functions**. It is a set of codes that perform a particular task. A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**.  **Syntax of method :**  <access\_modifier> <return\_type> <method\_name > ( list\_ of \_parameters ) { // body }

**## What is loop ? 🡪**  looping is also called as iterations. The process of repeatedly executing a statements and is called as looing. The statements may be executed multiple times. If a loop executing continuous then it called as infinite loop. In iteration statement, there are three types of loops: 1) **For loop**: The for loop is used when the number of iterations is known beforehand. It has three parts: initialization, condition, and increment/decrement. **Syntax:**  for (initialization; condition; increment/decrement) { // Code to be executed } **Example :**  Public class forexample { public static void main ( String [] args ) { for (int i=1; i<=10; i++ ) { System.out.println(i); } } } **While loop :**  The java while loop is used to iterated a part of the program several times. It the numbers of iteration is not fixed, it is recommended to use while loop. **Syntax:** while (condition) { // code to be executed } **Example:** Public class whileExample { public static void main ( String [] args ) { int i=1; while ( i<=10; i++ ) { System.out.println(i); i++; } } } **Do- while loop :**  The java do – while loop is used to iterate a part of the program several times . If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do – while loop.  **Syntax :**  Do { //code to be executed } while (condition ) ; **Example :** Public class doWhileExample { public static void main ( String [] args ) { int i=1; do { System.out.println(i); i++ } while(i<=8); } }

**## Call By methods :** To call a method in java, write the methods name followed by two parents () and a semicolon; **example:**  Public class main{ static void myMethod() { System.out.println(“ hello java “); } public static void main (string[] args ) { myMethod(); } }

**## java Arrays :**  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**: String[] cars; The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

**One Dimensional Array:**  it is a list of the variable of similar data types. It allows random access and all the elements can be accessed with the help of their index. The size of the array is fixed. For a dynamically sized array, vector can be used in C++. Representation of 1D array. **## Two Dimensional Array:**  it is a list of lists of the variable of the same data types. It also allows random access and all the elements can be accessed with the help of their index. It can also be seen as a collection of 1D arrays. It is also knowns as the matrix. Its dimension can be increased from 2 to 3 and 4 so on. They all are referred to as a multi-dimension array. The most common multidimensional array is a 2D array. Representation of 2D array . 

**Example of Array :**

**class** Testarray{   **public** **static** **void** main(String args[]){   **int** a[]=**new** **int**[5];//declaration and instantiation   a[0]=10;//initialization   a[1]=20;   a[2]=70;   a[3]=40;   a[4]=50; **for**(**int** i=0;i<a.length;i++) //length is the property of array   System.out.println(a[i]);   }

**## Rectangle array example:** public class Rectangle{   public static void main(String args[])   int width=5;   int height=10;  int area=width\*height;   System.out.println("Area of rectangle="+area);  }  }

**# Input / Output in java : 🡪** Programs read inputs from data source ( e.g. keyboard, file, network, memory buffer, or another program ) and write outputs to data sinks ( e.g. display console, file , network, memory buffer, or another program). Java I/O is used to process the input and produce the output. Java uses the concept of stream to make I/O operation fast. The java.io package is used for java i/o (file handling)

**## Multidimensional Array in Java : 🡪**In such case, data is stored in row and column based index (also known as matrix form). **Syntax to Declare Multidimensional Array in Java** dataType[][] arrayRefVar; (or)   dataType [][]arrayRefVar; (or) dataType arrayRefVar[][]; (or)  dataType []arrayRefVar[];    **Example to instantiate Multidimensional Array in Java** **int**[][] arr=**new** **int**[3][3];//3 row and 3 column   // Java program to take an integer // as input and print it class TakeInput { // main function public static void main(String[] args) { // Declare the variables int num; // Input the integer System.out.println("Enter the integer: "); // Create Scanner object Scanner s = new Scanner(System.in); // Read the next integer from the screen num = s.nextInt(); // Display the integer System.out.println("Entered integer is: "+ num); } }

**## Greater No. example in array:** public class GreaterNo { public static void main(String[] args) { if (20 > 18) { System.out.println("20 is greater than 18"); // obviously } } } public class GreaterNo { public static void main (String[] args ) { int x=20; int y=18; if (x > y ) { System.out.println(“x is greater than y” ); } } }

**## largest Element Array example:** public class LargestElement\_array {     public static void main(String[] args) {  //Initialize array    int [] arr = new int [] {25, 11, 7, 75, 56};   //Initialize max with first element of array.  int max = arr[0];    //Loop through the array   for (int i = 0; i < arr.length; i++) {     //Compare elements of array with  max      if(arr[i] > max)   max = arr[i];    }  System.out.println("Largest element present in given array: " + max);    }  }

**## Class :**  A class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:   
 **Modifiers** : A class can be public or has default access (Refer [this](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/) for details). **class keyword:**class keyword is used to create a class. **Class name:** The name should begin with a initial letter (capitalized by convention). **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface. **Body:** The class body surrounded by braces, { }.

**# String ;** The toString() method in Java is a pre-defined method in the Object class. It serves as the base class for all Java classes. It is used to retrieve a string representation of an object. Nevertheless, developers can override this method in their own classes to provide a customized string representation. **Example:**  public class StringExample { public static void main(String args[]) { String str = **new** String("example"); System.out.println(str); } }

**## Constructors :**  java allows objects to initialize themselves when they are crated constructors. **Properties :**  --Initializes an object immediately upon creation. --Same name as the class in which it resides and syntactically similar to a method. --Is called automatically after the object is created. --Does not have return type.

**Java default Constructors**: A constructor is called "Default Constructor" when it doesn't have any parameter. Syntax of default constructor: <class\_name>(){} //Java Program to create and call a default constructor   **class** Bike1{ Bike1() { System.out.println("Bike is created"); }   **public** **static** **void** main(String args[]) {   Bike1 b=**new** Bike1();  }   }

**## Methods Overriding :** 🡪 If subclass has the same method as declared in the parent class, it is known as method overriding. In other words, if subclass provides the specific implementation of the methods that has been provided by one of its parent class, it is knowns as Method Overriding. **Advantage of java Method Overriding:**  Provides specific implementation of a method that is already provided by its super class. **Rules for Method Overriding:**  method must have same name as in the parent class. Method must have same parameters as in the parent class. Must be IS-A relationship

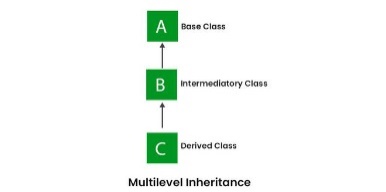
**Example: class** Parent { **void** show() { System.out.println("Parent's show()"); } } **class** Child **extends** Parent {   @Override **void** show() { System.out.println("Child's show()")    }

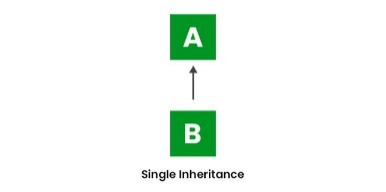
} **class** Main {  **public** **static** **void** main(String[] args)  {Parent obj1 = **new** Parent();  obj1.show();  Parent obj2 = **new** Child()  obj2.show();  } }

**## Finalize method** : 🡪It is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is not a reserved keyword.  
Once the finalize method completes immediately Garbage Collector destroy that object. finalize method is present in Object class and its syntax is: protected void finalize throws Throwable{}

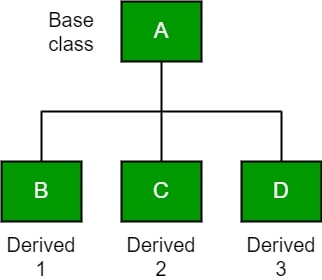
**## Method Overloading:** Method overloading in Java means having two or more methods (or functions) in a class with the same name and different arguments (or parameters). It can be with a different number of arguments or different data types of arguments.  *Also Read:*[*Data Types in Java – Primitive and Non-Primitive Data Types Explained*](http://shiksha.com/online-courses/articles/data-types-in-java-primitive-and-non-primitive-data-types) For instance: void function1(double a) { ... } void function1(int a, int b, double c) { ... } float function1(float a) { ...} double function1(int a, float b) { ... } In the above example, function1() is overloaded using a different number of parameters and different data types of parameters.

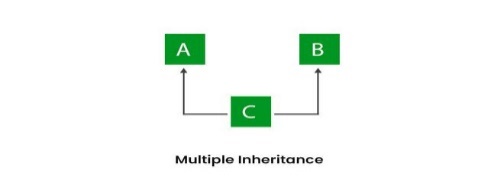
|  |  |
| --- | --- |
| **Overloading** | **Overriding** |
| Compile-time | Run-time |
| Increases the readability of the program. | Grants the specific implementation of the method already provided by its parent class or superclass. |
| Same name and different signatures | Same name and same signature |
| Return type can or can not be the same | Return type Must be the same or co-variant |
| Binding type is static | Binding type Dynamic |
| It performance good | Its performance poor |
| Arguments list should be same | Argument list should be different |

**## Inheritance:** When we construct a new class from existing class in such a way that the new class access all the features and properties of existing class called inheritance. Note: In Java, extends keyword is used to perform inheritance. It provide code reusability. We can’t access private members of class through inheritance. A sub class contains all the features of super class so, we should create the object of sub class. Method overriding only possible through inheritance. **Syntax:**  class parent { //code } class child extends parent { //code }

**Types of inheritance** : **1)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. Sometimes, it is also known as simple inheritance. In the below figure, ‘A’ is a parent class and ‘B’ is a child class. The class ‘B’ inherits all the properties of the class ‘A’. **Example:** **import** **java.io.\***; **import** **java.lang.\***; **import** **java.util.\***; **class** **One** { **public** void print\_geek() { System.out.println("Geeks"); } } **class** **Two** **extends** One { **public** void print\_for() { System.out.println("for"); } } **public** **class** **Main** { **public** **static** void main(String[] args) { Two g = **new** Two(); g.print\_geek(); g.print\_for(); g.print\_geek(); } } **Output :** Geeks , for , Geeks **2. Multilevel Inheritance** In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91).

**Example:** import java.io.\*; import java.lang.\*; import java.util.\*; class One { **public** void print\_geek() { System.out.println("Geeks"); } } **class** **Two** **extends** One { **public** void print\_for() { System.out.println("for"); } } **class** **Three** **extends** Two { **public** void print\_lastgeek() { System.out.println("Geeks"); } } **public** **class** **Main** { **public** **static** void main(String[] args) { Three g = **new** Three(); g.print\_geek(); g.print\_for(); g.print\_lastgeek(); } } **Output:**  Geeks , for , Geeks

**3. Hierarchical Inheritance** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived classes B, C, and D. 

**4. Multiple Inheritance (Through Interfaces)** In [Multiple inheritances](https://www.geeksforgeeks.org/java-and-multiple-inheritance), one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support [multiple inheritances](https://www.geeksforgeeks.org/java-and-multiple-inheritance) with classes. In Java, we can achieve multiple inheritances only through [Interfaces](https://www.geeksforgeeks.org/interfaces-in-java). In the image below, Class C is derived from interfaces A and B.

**## Abstract Methods and Classes:** Data abstraction is the process of hiding certain details and showing only essential information to the user.. The abstract keyword is a non-access modifier, used for classes and methods: **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). **Abstract method:**can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from). **Points to Remember** --An abstract class must be declared with an abstract keyword. --It can have abstract and non-abstract methods. --It cannot be instantiated. --It can have constructors and static methods also. --It can have final methods which will force the subclass not to change the body of the method. **Example:**  abstract class Bike{ abstract void rum(); class Honda extends Bike { void run() { System.out.println(“running safely..” ); } public static void main (String args[] ) { Bike obj=new Honda(); obj. run(); } }

**# Define Interface:** Interface is just like a class, which contain only abstract method. An Interface in Java is a blueprint of a class. An Interface is declared by using the interface keyword. Interfaces are used to implement abstraction **Use of interface**: There are mainly three reasons to use interface. They are given below. --It is used to achieve abstraction. --By interface, we can support the functionality of multiple inheritance. --It can be used to achieve loose coupling. **Example:**  interface itemconstants { int code=1001; string name= “fan”; void display (); }

**# Java Package & API :** A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories: --Built-in Packages (packages from the Java API) -- User-defined Packages (create your own packages) **Built-in Packages** The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment. The library contains components for managing input, database programming, and much more. The complete list can be found at Oracles website: <https://docs.oracle.com/javase/8/docs/api/>. The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package. **Syntax** : import package.name Class; import package.name.\*;

**## Exceptions Handling in java :** Exception Handling in Java is one of the effective means to handle runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**## Why Exception Occurs?**  🡪 An exception can occur for many different reasons, below given are some scenarios where exception occurs. A user has entered invalid data. A file that needs to be opened cannot be found. A network connection has been lost in the middle of communications or the JVM has run out of memory. **# Types of Program errors :** We distinguish between the following types of errors: **Syntax error:**  Errors dur to the fact that the syntax of the language is not represented . **Semantic error :**  Errors due to an improper use of program statements. **Logical error:**  Errors due to the fact that the specification is not respected. From the point of view of when errors are deleted.  **Compile time error:**  Syntax error & Static error indicated by the compiler.

|  |  |
| --- | --- |
| **Call by value** | **Call by reference** |
| Simple types arguments are passed to method | Object are passed to methods |
| This method copies the value of an argument into the formula parameter of the subroutine. | In this method, reference to an argument is passed the parameter |
| Doesn’t access actual argument | This reference is used to access the actual argument |
| Thus change made to parameter of the subroutine have no effect on the argument | Thus change made to parameter will have an effect on the argument |

**## Getters And Setters :** Getter and setters are used to effectively protect your data, particularly when creating classes. For each variable, the get method returns its value, while the set method sets the value. Getters start with get, followed by the variable name, with the first letter od f the variable name capitalized. Setters start with set, followed by the variable name, with the first letter of the variable name capitalized. **Example:**  public class Vehicle { private String color; // Getter public String getColor() { return color; } //Setter public void setColor(String c) { this.color=c;} } The getter method returns the value of the attribute. The setter method takes a parameter and assigns it o the attribute.

**## Throw keyword :**  -- Keyword is used to explicitly throw an exception/ custom exception. throw new ExceptionName(“Error Message”); --Throw either checked or unchecked exception . throw net ThrowableInstance ThrowableInstance must be an object of type Throwable / subclass Throwable -- There are two ways to obtain a Throwable objects: -using a parameter into a catch clause -Creating one with the new operator

**Throw keyword Example :** public class bank { public static void main (String args[]) { int balance =100, withdraw=1000; if balance < withdraw ) { // ArithwticException e= new ArithmeticException (“No money please”); //throw e ; //throw new ArithmeticException(“No money”) ; } else { System.out.println(“Draw & enjoy sir, Best wishes of the day”); } } }

|  |  |
| --- | --- |
| **Throw keyword** | **Throws keyword** |
| Throw is used to explicitly throw an exception. | Throws is used to declare an exception |
| Checked exception can not be propagated without throws | Checked exception can be propagated with throws |
| Throw is followed by an instance | Throws is followed by class |
| Throw is used within the method | Throws is used with the method signature |
| You cannot throw multiple exception | You can declare multiple exception |

**## what is Thread ?** 🡪 A thread is a single sequential flow of control within a program. Thread does not have its own address space but use the memory and other resources of the process in which it executes. There may be several threads in one process. The java virtual machine manage there and schedules them for execution. **Syntax:**  public class Main extends Thread { public void run() { System.out.println("This code is running in a thread"); } } **## Streams :** java implements streams within class hierarchies defined in the java.io package. A stream is an order sequence of data. A stream is linked to a physical device by the java I/O system. All streams behave in the same manner, even if the actual physical devices to which they are linked differ. An I/O stream represents an input source or an output destination.

**# Multithreading** : Multithreading in java is a process of executing multi processes simultaneously . A program is divided into two or more subprograms, which can be implemented at the same time in parallel. Multiprocessing and multithreading, both are used to achieve multitasking. Java Multithreading is mostly used in games animation etc. **Advantage: --**it doesn’t block the user. --Can perform many operations together so it saves time. --Threads are independent so it doesn’t affect other threads.

**# Java Synchronization:**  Generally threads use their own data and methods provided inside their run () methods. But if we wish to use data and methods outside the thread’s run() method, they may compete for the same resource and may lead to serious problems. Java enables us to overcome this problem using a technique know as Synchronization . for ex. : One thread may try to read a record from a file while another is still writing to the same file. When the method declared as synchronized , java creates a “ monitor” and hands it over to the thread that calls the method first time. Synchronized (look- object ) { ………// code here is synchronized }

**# I/O Streams:**  A stream can represent many different kinds of source and destinations. Disk files, devices, other programs, a network socket, and memory arrays. Stream support many different kinds of data – simple bytes, primitive data types, localized characters, and objects . Some streams simply pass on data; other manipulate and transform the data in useful ways. **I/O stream categories :** --Data Streams –Processing streams --Input streams –output streams – Character streams – Byte stream **Input stream :** A program uses an input stream to read data from a source one item at a time. **Output Stream :**  A program uses an output stream to write data to a destination one item at time.

**#Java Applet : 🡪**A **Java Applet** is a small application that is written in the Java programming language and can be embedded into a web page. Applets are executed in a web browser using a Java Virtual Machine (JVM). They were typically used to provide interactive features on websites, such as games, animations, or tools that run directly within a browser. Java Applets have largely fallen out of use due to security concerns, the rise of other web technologies like HTML5, JavaScript, and CSS, and the decreasing support for Java in modern web browsers. **Advantage of Applet** There are many advantages of applet. They are as follows: --It works at client side so less response time. –Secured --It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc. **Drawback of Applet** --Plugin is required at client browser to execute applet

**## HTML < applet >** : HTML <applet> tag was used to embed the Java applet in an HTML document. This element has been deprecated in HTML 4.0 and instead of it we can use <object> and newly added element <embed>. The use of Java applet is also deprecated, and most browsers do not support the use of plugins.  **Syntax** <applet code="URL" height="200" width="100">.............</applet>   **Example** <!DOCTYPE html>   <html>  <head>   <title>Applet Tag</title>    </head>    <body>     <p>Example of Applet Tag</p>     <applet code="Shapes.class" align="right" height="200" width="300">    <b>Sorry! you need Java to see **this**</b>    </applet>   </body>   </html>

 ## **Protected keyword** :

🡪The **protected keyword** in Java refers to one of its access modifiers. The methods or data members declared as protected can be accessed from --Within the same class. --Subclasses of the same packages . --Different classes of the same packages. --Subclasses of different packages.

**Example :** class Parent { protected void display() { System.out.println("This is a protected method."); } } class Child extends Parent { public static void main(String[] args) { Child obj = new Child(); obj.display(); // Accessible because Child is a subclass of Parent } }

**##**  **I/O stream Example:** import java.io.FileInputStream; import java.io.FileOutputStream; import java.io.IOException; public class SimpleIOExample { public static void main(String[] args) { FileInputStream inputStream = null; FileOutputStream outputStream = null; try { inputStream = new FileInputStream("input.txt"); outputStream = new FileOutputStream("output.txt"); int data; while ((data = inputStream.read()) != -1) { outputStream.write(data); } System.out.println("File copied successfully!"); } catch (IOException e) { e.printStackTrace(); } finally { try { if (inputStream != null) { inputStream.close(); } if (outputStream != null) { outputStream.close(); } } catch (IOException e) { e.printStackTrace(); } } } }

**## Life Cycle of a Thread**

🡪There are multiple states of the thread in a lifecycle as mentioned below: **1) *New Thread:****When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.* **2) *Runnable State:****A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.   
A multi-threaded program allocates a* ***3) Blocked:****The thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread. The thread will move from the blocked state to runnable state when it acquires the lock.* ***4) Waiting state:****The thread will be in waiting state when it calls wait() method or join() method. It will move to the runnable state when other thread will notify or that thread will be terminated.* ***5) Timed Waiting:****A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.*