Q) HTML ?

HTML, which stands for HyperText Markup Language, is the fundamental building block for web pages. It defines the structure and content of webpages, acting like a blueprint that web browsers use to render the information and visuals you see online.

HTML uses a system of tags to define the different elements on a webpage. These tags are written within angle brackets (< and >). Each tags has a specific meaning

Q) HTML Tags?

HTML tags are the building blocks of web pages. They define the structure and content of a webpage, telling the browser how to display the information. Imagine an HTML tag like a container with a label. The opening tag < defines the start of the container, the closing tag > defines the end, and the text inside describes the content within the container

- **Heading tags (`<h1>` to `<h6>`):** Define headings of different sizes. `<h1>` is the most prominent heading, and `<h6>` is the least.
- **Paragraph tag (`<p>`):** Defines a paragraph of text.
- **Anchor tag (`<a>`):** Creates a hyperlink to another webpage or section of the current page.
- **Image tag (`<img>`):** Inserts an image into the webpage.
- **List tags (`<ul>` for unordered lists, `<ol>` for ordered lists):** Create bulleted or numbered lists.
- **Table tags (`<table>`, `<tr>`, `<td>`):** Define tables with rows and columns for structured data.

Link and Hyper Link

A link is simply a connection between two different points, whereas a hyperlink is a type of link that uses HTML code which when clicked on will direct the user to another internet page.

Q)Attributes?

HTML attributes are special keywords that provide additional information about HTML elements. They act like modifiers that define the element's behavior or characteristics.

Link: a link (more precisely called a hyperlink) is an element that directs users to a different web page or a specific section within the current page when clicked

Q) identifers in html

identifiers is a unique label assigned to a specific element. This identifier allows you to target and manipulate that element using CSS and JavaScript.

ID and Class are identifiers

Q) Styling ?

HTML itself isn't responsible for styling a webpage. HTML defines the structure and content, but how it looks is controlled by Cascading Style Sheets (CSS)

1. **Inline Styles:** You can use the **`style`** attribute directly within HTML tags to apply specific styles. This isn't generally recommended for large-scale styling because it can make your code cluttered and hard to maintain.

```
<h1 style="color: blue; font-size: 20px;">This is a heading</h1>
```

2. **Internal Styles:** You can define styles within the <**`head`**> section of your HTML document using the <**`style`**> tag.

```
<head><style>
h1 {
 color: blue;
 font-size: 20px;
}
</style></head>
<body>
 <h1>This is a heading</h1>
</body>
```

3. **External CSS**

This involves creating a separate **`.css`** file containing all your styles and linking it to your HTML document using the <**`link`**> tag in the <**`head`**> section.

```
<link rel="stylesheet" href="mystyle.css">
```

Q) Link and Href?

- Use the <link> tag to link external resources (like CSS) to your HTML document.

- Use the href attribute within the <link> tag to specify the URL of the external resource.
- Use the href attribute within the <a> tag to specify the destination URL of a hyperlink.

Q) DIV tag

The `<div>` tag creates a division. Its has opening tag and closing tag.

Its primary function is to group elements together and provide a way to apply styles (using CSS) to those elements as a whole.

Think of it as a container or box that holds other HTML elements like headings, paragraphs, images, lists, etc.

Uses - Grouping related content, Creating sections:, Applying styles.

Q) Anchor tag

The anchor tag, also known as the `<a>` tag, is a fundamental element in HTML used to create hyperlinks. These hyperlinks allow users to navigate between different webpages, sections within the same webpage, or even trigger actions like downloading a file or sending an email.

Q) Heading Tag

Heading tags, denoted by `<h1>` to `<h6>` in HTML, are used to structure the content of your webpage and define different levels of importance for headings.

H1 to H6

Q) Span tag

The tag is an inline container used to mark up a part of a text, or a part of a document.

HTML Table – Colspan:
To make a cell span over multiple columns, use the colspan attribute

HTML Table – Rowspan:
To make a cell span over multiple rows, use the rowspan attribute

Forms – action, target, methods

MySQL & SQL

MySQL is a specific relational database management system (RDBMS) that implements the SQL language. MySQL stores data in a structured format using tables with rows and columns. It provides the underlying engine to create, manage, and access databases.

SQL is a standardized programming language specifically designed for managing relational databases. you can use SQL to interact with relational databases in various ways, including:

Inserting data, Updating data, deleting dta etc…

What is indexing in sql?

Q)Constraints of SQL ?

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

Q) Primary key and foreign key

PRIMARY KEY:

- A primary key uniquely identifies each row in a table. It's a combination of one or more columns that acts as the main identifier for each record. A primary key enforces both uniqueness (no duplicates) and NOT NULL (no missing values) for the columns it includes.

FOREIGN KEY:

- This constraint establishes a link between two tables. It references a primary key in another table, ensuring data consistency between related tables. For example, an `order` table might have a foreign key referencing a `CUSTOMER` table's primary key, linking specific orders to individual customers.

Q)Normalization

Normalization in SQL is a process of organizing your database tables to minimize data redundancy, improve data integrity, and enhance overall database design. It involves structuring your tables to efficiently store and retrieve data while reducing the chances of errors and inconsistencies.

- **First Normal Form (1NF):** The most basic level. It ensures each table cell contains a single atomic value (indivisible unit of data) and eliminates duplicate rows within a table.
- **Second Normal Form (2NF):** Complies with 1NF and additionally eliminates partial dependencies. In simpler terms, this means a column shouldn't depend on only a part of the primary key.
- **Third Normal Form (3NF):** Follows 2NF and removes transitive dependencies. This ensures that every non-key attribute (column) in a table has a direct dependency on the entire primary key, not just a part of it.

Q)Types of SQL Commands

There are five types of SQL commands:

● DDL

● DML

● DCL

● TCL

● DQL.

Data Definition Language (DDL)

● These commands are used to define the structure of your database. like creating a table, deleting a table, altering a table

Eg. CREATE, ALTER, DROP, TRUNCATE.

Data Manipulation Language (DML)

● DML commands are used to modify the database. It is responsible for all forms of changes in the database.

Eg. INSERT, UPDATE, DELETE.

Data Query Language (DQL)

● DQL is used to fetch the data from the database.

Eg: SELECT

SELECT: This is the same as the projection operation of relational algebra. It is used to

select the attribute based on the condition described by WHERE clause.

Data Control Language (DCL)

- **GRANT**: Used to give users access privileges to the database.
- **REVOKE**: Used to take back permissions from users.

Transaction Control Language (TCL)

- **COMMIT**: Used to save all changes made during the current transaction.
- **ROLLBACK**: Used to undo transactions that have not yet been saved to the database.
- **SAVEPOINT**: Used to set a point within a transaction to which you can later roll back.
- **SET TRANSACTION**: Used to specify characteristics for the transaction.

Joins

Inner join

- This is the most common type of join. It selects rows from two tables where the joining columns have matching values.
- Only rows with matches in both tables are included in the result set.

SELECT * FROM customers INNER JOIN orders ON customers.customer_id = orders.customer_id;

Left join

- This join includes all rows from the left and matching rows from the right table.
- If there's no match in the right table for a particular row in the left table, the corresponding columns from the right table will contain null values.

SELECT * FROM customers LEFT OUTER JOIN orders ON customer.customer_id = order.customer_id;

Right Join

- Similar to a left join, but it includes all rows from the right table and matching rows from the left table.
- Missing matches from the left table will result in null values in the corresponding left table columns.

SELECT * FROM customers RIGHT OUTER JOIN orders ON coustmer.customer_id = order.customer_id;

Full Joins

- This join combines all rows from both tables, regardless of whether there's a match in the joining columns.
- Missing matches in either table will result in null values in the corresponding columns.

SELECT * FROM customers FULL OUTER JOIN orders ON customers.customer_id = order.customer_id;

UPDATE:

The UPDATE statement is used to modify existing records in a table

UPDATE employees SET salary = 50000 WHERE department = 'HR';

Delete:

The DELETE statement is used to remove one or more rows from a table based on a specified condition in the WHERE clause

DELETE FROM table_name WHERE condition;

TRUNCATE:

The TRUNCATE statement is used to remove all rows from a table quickly and efficiently.

TRUNCATE TABLE table_name;

DROP:

The DROP statement is used to remove database objects like tables, views, indexes, or even databases themselves. It removes both the structure and data

DROP TABLE products;

Limit and OFFset

JAVA Questions

1) What are the features of java?

Object-Oriented: Java is fully object-oriented, meaning programs are built around objects that encapsulate data and behavior. This leads to modular, reusable, and maintainable code.expand_more

Platform Independent: Java's "write once, run anywhere" motto stems from its bytecode compilation.expand_more Java code compiles into bytecode, which runs on any machine with a Java Virtual Machine (JVM), regardless of the underlying operating system. Expand more

Secure: Java's memory management, automatic garbage collection, and lack of pointers make it inherently secure compared to languages like C++. Additionally, the bytecode sandbox and access modifiers further enhance security.

Robust: Java's exception handling, strong typing, and garbage collection contribute to its robustness.expand_more These features help prevent errors and memory leaks, making Java applications reliable.expand_more

Multithreaded: Java supports multithreading, allowing applications to handle multiple tasks concurrently. expand_more This is essential for building responsive and efficient applications.expand_more

Portable: Bytecode portability enables Java applications to run on various platforms without modification.expand_more This simplifies deployment and maintenance across different environments.

2) Explain JRE, JDK, JVM

JDK is the complete package for developing java application. It contains JRE & development tools.
JRE provides the environment for the code execution. JRE is the part of JDK.JRE contains JVM, library set and other files..
JVM is primarily responsible for running the java application. It is part of JRE.

3) What is java Package?

Package is a set of related classes and interfaces. Packages in java can be categorised in two forms, built-in packages and user-defined packages. There are many built-in packages such as java.lang, java. io, java.util, java.sql etc. Packages are just similar to folders in the disk.

4) What is constructor. what is the rules for creating constructor. Types of constructor

A constructor is a special type of method with the same name as the class and **no return type**. Its primary purpose is to **initialize objects** when they are created using the NEW.

Rules

- Must match the class name exactly.
- No return type (not even void).
- Can be declared with any access modifier (public, private, protected, or default).
- Can have zero or more parameters to receive initial values for the object's fields.
- If you don't explicitly define a constructor, Java provides a default no argument constructor that assigns default values to the fields.

Types of constructors

- **Default constructor:** Java provides a default no argument constructor that assigns default values to the fields, provided by Java if you don't define any.
- **Parameterized constructor:** Takes input parameters to initialize the object's fields with specific values.
- **Private constructor:** Makes the constructor accessible only within the class, preventing object creation outside the class.

5) What is access modifier?

In Java, access specifiers are the keywords used before a class, method or a

variable name which defines the access scope. The access specifiers are:

1. **public**: class, method, field is accessible from anywhere.
2. **protected**: method, field can be accessed from the all the classes in the same package and also from subclasses in different packages.
3. **default**: class, method ,field can be accessed only from the same package and not from outside the package.(If you not use any access modifiers, it will be treated as default access modifier.)
4. **private**: method, field can be accessed from the same class to which they belong to. (If a variable is declared as private, we cannot access it from outside the class. So, we use public methods like setters and getters for accessing this variable. Such classes are called **fully encapsulated class**.)

6) what is data type?

Java uses data types to define the type and size of values that can be stored in variables.

Primitive Data Types: These are fundamental data types built into the language itself.
Char, Boolean, double, float, long, int, short, byte

Non-Primitive Data Types: These are user-defined or predefined data types that are not fundamental to the language. They include:

- **String:** Represents a sequence of characters.
- **Arrays:** Ordered collections of elements of the same data type.
- **Classes:** Complex data structures with attributes (data) and methods (functions).
- **Interfaces:** Define a set of methods that classes can implement.

7) what is type casting, diff type of typecasting .

Type casting is the process of explicitly converting a value from one data type to another

```
// Widening conversion (automatic)
int age = 25;
long largeAge = age; // No casting needed, automatic conversion

// Narrowing conversion (manual)
```

```
double pi = 3.14159;
int roundedPi = (int) pi; // Data loss, decimal part truncated
```

8) what is return type ?

A return type defines the **kind of data** a method is expected to produce when it finishes execution. It essentially tells the caller what value the method will provide back after completing its tasks.

- **Primitive Data Types:** Methods can return basic data types like int, double, boolean, char, etc.
- **Non-Primitive Data Types:** Methods can return objects of classes, arrays, interfaces, etc.
- **Void:** If a method doesn't return any value, its return type is void.

9) what is variables and its type?

variables are named memory locations that store **data** during program execution. Each variable has a **data type**, which determines the kind of data it can hold and the operations that can be performed on it.

- **Local Variables:** Declared within methods and exist only during the execution of that method.
- **Instance Variables:** accessible throughout the entire class, but specific to each object (instance) of the class.
- **Static Variables:** Accessible to all instances of the class and even without creating an object of the class.

10) what is operator and its type?

Operators are symbols that perform **operations** on operands (values or variables). They act like tools that manipulate data and control program flow.

1. **Arithmetic Operators:** Used for mathematical calculations like addition, subtraction, multiplication, division, and modulus:
   - +, -, *, /, %
2. **Relational Operators:** Compare two values and return a boolean result (true or false):
   - == (equal to), != (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to)
3. **Logical Operators:** Combine boolean values using and, or, not:
   - && (and), || (or), ! (not)

4. **Assignment Operators:** Assign values to variables, with variations like shorthand assignments:
    - =, +=, -=, *=, /=, %=
5. **Bitwise Operators:** Operate on individual bits of data for low-level operations:
    - & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift)
6. **Increment/Decrement Operators:** Increase or decrease a variable by 1:
    - ++, -- (pre-increment/decrement and post-increment/decrement)
7. **Ternary Operator:** Conditional expression offering a compact alternative to if-else statements:
    - condition ? valueIfTrue : valueIfFalse
8. **Instanceof Operator:** Checks if an object belongs to a certain class or implements an interface:
    - object instanceof class/interface

11) What is method and its type?

A **method** is a self-contained block of code designed to perform a specific task or operation. It helps organize and modularize your code, improving readability, reusability, and maintainability.

Types of Methods:

1. Predefined Methods:

- These are built-in methods available in Java libraries and classes, like System.out.println() for printing output.
- You don't need to define them, but simply use them with relevant syntax.

2. User-Defined Methods:

- You create these methods yourself to encapsulate specific functionalities within your code.
- They offer flexibility and customization based on your program's needs.

12) What is public static void main(String Args[])?

Public: Access specifier

Static: You can call it directly using the class name, without needing an object instance.

Void: return type

Main: method name.

String Args: This parameter is an array of String objects.

13) what is call by value and call by reference?

Call by Value:

When a function is called using call by value, a **copy** of the argument's value is passed to the function. This means any changes made within the function to the passed value **do not affect the original variable** outside the function.

Call by Reference:

Instead of copying the value, the function receives a **reference** to the memory location of the original variable. This means any changes made inside the function **directly modify the original variable**.

14) what is recursion?

Recursion in Java is a programming technique where a **function calls itself directly or indirectly**

15) what is scanner class?

The Scanner class in Java is a tool for **reading and parsing input**, most commonly used for getting user input from the console or reading data from a file.

16) what is statements and its type?

statements are fundamental building blocks of java code. They represent complete instructions that the compiler translates into actions for the computer to execute. Each statement ends with a semicolon (;)

Expression Statements:

- Perform calculations or assignments and often return a value.

Eg: `sum = x + y;`

Declaration Statements:

- Introduce new variables and allocate memory for them.

Eg: int age = 30;

Control Flow Statements:

- Change the execution flow of the program based on conditions.

Eg: if (age >= 18) { vote(); }

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

Return Statements:

- Used within methods to return a value to the calling code.

17) Explain java keywords(this, super, new) -> 68 keywords

A Java keyword is a **reserved word** that has a specific meaning within the Java programming language. These words cannot be used as names for variables, methods, or classes because they already have defined functions within the language itself.

This: Refers to the **current object** itself within a method or constructor.

- Accessing instance variables of the current object: this.age = 25;
- Calling other methods of the current object:

Super: Refers to the **immediate parent class** of the current class.

- Calling methods of the parent class: super.printMessage();
- Accessing instance variables of the parent class:

New: Used to **create a new instance** of a class or object.

Final: Once assigned a value, the variable cannot be reassigned another value.

- Prevents subclasses from overriding the method in their own implementations.
- Prevents other classes from being extended and inheriting its functionality.

18) what is enhanced for loop(for each loop)

The enhanced for loop, also known as the **for-each loop**, is a way to simplify iterating over arrays and collections

Syntax: for (data_type variable : collection)

19) **What is upcasting?**

If the reference variable of Parent class refers to the object of Child

class, it is known as upcasting.

Syntax
class A{}
class B **extends** A{}

A a=**new** B ();//upcasting

20) What is method over loading and method over ridding?

Method Overloading: This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters

Method Overridding

Creating a method in a **subclass** that has the **same name, parameter list, and return type** as a method in its **superclass**.

Strings

1) what is a string and how to create a string?

A **string** is a sequence of characters, typically used to represent text data. These characters can include letters, numbers, symbols, and spaces. Enclosed in double quotes.

- Strings are **immutable** in Java, meaning you cannot modify the individual characters within an existing string.
- If you need to modify the content, you have to create a new string with the desired changes.
- Java also provides several methods for working with strings, such as length(), substring(), and indexOf().

1. Using string literals: Enclose the desired text within double quotes (").

- Example: String greeting = "Hello, world!";

2. Using the new **keyword:**

- Create a new String object using the new keyword and the String class constructor.
- Example: String name = new String("John Doe");

2) **what are the string comparison method?**

1. equals(String anotherString):

- It compares the **content** of the strings, meaning individual characters must match exactly in the same order for them to be considered equal.
- Case matters, so "Hello" and "hello" are different.
- Returns true if the strings are equal, false otherwise.

2. equalsIgnoreCase(String anotherString):

- Similar to equals, but **ignores case sensitivity**.
- "Hello" and "hello" would be considered equal with this method.
- Useful when comparison shouldn't be affected by case differences.

3. compareTo(String anotherString):

- Compares strings **lexicographically** (dictionary order).
- Returns a negative integer if the string is less than the other string, 0 if they are equal, and a positive integer if it's greater.
- Considers both characters and their Unicode values for comparison.
- Can be useful for sorting or ordering strings.

4. == Operator:

- **Compares memory addresses:** Checks if two object references point to the same **object instance** in memory.
- **Primitive types vs. Objects:**
    - For primitive types (like int, double, boolean), == compares the actual values.
    - For objects, == compares memory addresses, not object content.

3) **Difference between String, String builder, string buffer?**

String:

- **Immutable:** Once created, the content of a String object cannot be changed. Any attempt to modify it will create a new String object.
- Simple string operations like printing, concatenation, and comparisons where content needs to remain constant.

StringBuilder:

- **Mutable:** You can modify the content of a StringBuilder object after its creation.
- Building strings dynamically, concatenating multiple strings, and modifying string content efficiently in single-threaded environments.
- **Less efficient**

```
StringBuilder str = new StringBuilder();
```

StringBuffer:

- **Mutable:** Same as StringBuilder, you can modify the content after creation.
- **More efficient**
- String manipulation in multithreaded environments where thread safety is crucial, even if it compromises some performance compared to StringBuilder
- Both StringBuilder and StringBuffer offer similar methods for manipulating strings.
- The Java String class internally uses StringBuffer or StringBuilder for efficient concatenation operations.

Array

1) **what is array. its advantages and disadvantages?**

An **array** is a data structure in Java used to store a collection of elements of the **same data type**. It's like a fixed-size container where each element has a unique index (starting from 0) to access it.

Advantages:

- **Efficient access:** Accessing individual elements using their index is very fast due to direct memory allocation.
- **Simple and familiar:** The concept of arrays is intuitive and easy to understand.
- **Memory efficient:** For storing large collections of the same data type, arrays can be more memory-efficient than other data structures like linked lists.
- **Built-in functionalities:** Java provides various methods for array manipulation, like sorting, searching, and copying.

Disadvantages:

- **Fixed size:** Once created, the size of an array cannot be changed. This can be a limitation if you don't know the exact number of elements you need to store initially.
- **Memory waste:** If you don't use all the elements in an array, there can be wasted memory allocation.
- **Inefficient insertion and deletion:**

2) **Explain diff types of array?**

1. Single-dimensional Arrays:

- These are the most common type of arrays, storing a collection of elements of the same data type in a single row.
- Each element has an index, starting from 0, which allows you to access it directly.
- Example: int[] numbers = {1, 2, 3, 4, 5};

2. Multidimensional Arrays:

- These are essentially arrays of arrays, allowing you to represent data in a two-dimensional or higher grid-like structure.
- Each element is accessed using multiple indices, one for each dimension.
- Example: int [] [] matrix = {{1, 2, 3}, {4, 5, 6}};

3. Jagged Arrays: These are multidimensional arrays where each row can have a different length.

Example: int[][] jaggedArray = {{1, 2}, {3, 4, 5}};

3) **what is Declaration, Instantiation, and Initialization of Array?**

1. Declaration:

- This stage formally defines the existence of an array variable, specifying its **name** and **data type** of its elements.
- It doesn't allocate memory for the array itself, just a reference to where it will be stored later.

- Syntax: dataType[] arrayName; (e.g., int[] numbers;)

2. Instantiation:

- This step actually **creates the array** in memory, allocating the necessary space to store its elements.
- It uses the new keyword to allocate memory and specify the **size** of the array (number of elements it can hold).
- Syntax: datatype [] arrayName = new dataType[size]; (e.g., int numbers = new int[5];)

3. Initialization:

- This stage involves **assigning values** to the individual elements of the array.
- You can either initialize all elements individually or use specific syntax for collective initialization.

4) what is anonymous array?

An anonymous array is an array created **without** giving it a name. They are useful for one-time use scenarios where you don't need to refer to the array again later.

String and Array

Array:

- **Definition:** A data structure that holds a collection of **elements of the same data type**.
- **Data type:** Can store any data type (int, float, double, objects, etc.).
- **Size:** Fixed size upon declaration. Cannot be resized later.
- **Mutability:** Elements can be changed individually.
- **Memory:** Stored contiguously in memory for efficient access.
- **Example:** int[] numbers = {1, 2, 3};

String:

- **Definition:** An **object** representing a sequence of **characters**.
- **Data type:** String (a wrapper class for char arrays).

- **Size:** Technically fixed after creation, but can be indirectly modified using various methods (e.g., concat, substring).
- **Mutability:** Immutable. Modifications create a new String object.
- **Memory:** Stored in the String Constant Pool or heap depending on creation method.
- **Example:** String name = "Hello";

Opps Concepts

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

Object

● Any entity that has state and behaviour is known as an object. It can be physical or logical.
● An Object can be defined as an instance of a class.
● An object contains an address and takes up some space in memory.
● **Example:** A dog is an object because it has states like colour, name, breed, etc. as well as behaviours like wagging the tail, barking, eating, etc.

Class

● Collection of objects is called class.
● It is a logical entity.
● A class can also be defined as a blueprint from which you can create an individual object.
● Class doesn't consume any space.

Abstraction

● Abstraction is a process of hiding the implementation details and showing only functionality to the user.
● Another way, it shows only essential things to the user and hides the internal details.
● For example, sending SMS where you type the text and send the message.
● You don't know the internal processing about the message delivery.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

● A class which is declared as abstract is known as an **abstract class**.
● It needs to be extended and its method implemented.

Abstract Method:

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example

```
abstract class Birds
{
Birds() // Default Constructor
{
System.out.println(“Birds is created”);
}
abstract void fly(); // Abstract Method
void birdquality() // Non-abstract Method
{
System.out.println(“The eagles are a symbol of beauty, bravery,
courage, honour, pride, determination, and grace. “);
}
}

class Eagles extends Birds
{
void fly()
{
System.out.println("Eagles are high flyers");
}
public static void main(String args[])
{
Birds b = new Eagles();
b.fly();
b.birdquality();
}
}
```

2)Interface

● An interface is a blueprint of a class.
● It has static constants and abstract methods.
● The interface in Java is *a* mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
● Java Interface also represents the IS-A relationship**.**
● It cannot be instantiated just like the abstract class.
● Since Java 9, we can have private methods in an interface.

Reasons to use interface
● It is used to achieve abstraction.
● By interface, we can support the functionality of multiple inheritance.
● It can be used to achieve loose coupling.
How to declare an interface?
● An interface is declared by using the interface keyword.
● It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
● A class that implements an interface must implement all the methods declared in the interface.
Syntax:

```
interface < interface_name >{
// declare constant fields
// declare methods that abstract
// by default.
}
```

```
interface Printable
{
void print();
}
interface Showable
{
void show();
}
class A7 implements Printable,Showable
```

```
{
public void print()
{
System.out.println("Hello");
}
public void show()
{
System.out.println("Welcome");
}
public static void main(String args[])
{
A7 obj = new A7();
obj.print();
obj.show();  } }
```

Encapsulation

● Encapsulation is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.
● In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found
● We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
● The Java Bean class is the example of a fully encapsulated class.

Example
Student.java

```
public class Student
{
//private data member
private String name;
//getter method for name
public String getName()
{
return name;
}
//setter method for name
public void setName(String name1)
{
Name1=name
```

```
}
}
```

Test.java

```
class Test
{
public static void main(String[] args)
{
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

Polymorphism

- Polymorphism is a concept by which we can perform a **single action in different ways**.
- Polymorphism is derived from 2 Greek words: **poly and morphs**.
- The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Two types of polymorphism

- Runtime polymorphism
- Compile Time polymorphism.

Syntax

```
class A{}
class B extends A{}
A a=new B ();//upcasting
```

Runtime Polymorphism with Method

In runtime polymorphism, the decision of which method to call is **made at runtime** based on the actual object type. This is achieved through **method overriding** and **inheritance**.

Example

```
class Fruits
{
int speed limit=90;
void flavour()
{
System.out.println("Sweet/Sour/Bitter");}
}

class Apple extends Fruits{
void flavour()
{
System.out.println("Flavour is sweet");
}
public static void main(String args[])
{
int speed limit=150;
Fruits obj = new Apple(); //upcasting
obj.flavour();
System.out.println(obj.speed limit);
}
}
```

Eg:

Compile Time polymorphism

In compile-time polymorphism, the decision of which method to call is **made by the compiler at compile time** based on the method signature (name, parameters, return type). This is achieved through **method overloading**.

```
class SampleAdd
{
int add(int a, int b)
{
return a+b;
} int add(int a, int b,int c)
{
return a+b+c;
}
}

class Demo
{
```

```
public static void main(String args[])
{
SampleAdd obj=new SampleAdd();
System.out.println(obj.add(10,20));
System.out.println(obj.add(10,20,30));
}
}
```

Inheritance

- **Inheritance** is a mechanism in which one class acquires the property of another class.
- With inheritance, we can reuse the fields and methods of the existing class.
- Hence, inheritance facilitates Reusability and is an important concept of OOPs.
- A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Single Inheritance:

In Single Inheritance one class extends another class (one class only).

Class A extends Class B

```
class Parent
{
void Parent_Details()
{
System.out.println("Parent Details...");
}
} class Child extends Parent {
void Child_Details() {
System.out.println("Child Detail...");
}
}
```

```
public class SingleInheritance
{
public static void main(String args[])
{
Child c = new Child();
c.Parent_Details();
c.Child_Details();
}
}
```

Multilevel Inheritance:

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.

Class parent
Class child1 extends parents
Class child 2 extends child1

```
class Parent
{
void Parent_Details()
{
System.out.println("Parent Details...");
}
}

class Child1 extends Parent
{
void Child1_Details()
{
System.out.println("Child1 Detail...");
}
}

class Child2 extends Child1 {
void Child2_Details() {
System.out.println("Child2 Detail...");
}
}
public class MultilevelInheritance
{
```

```
public static void main(String args[]) {
Child2 c = new Child2();
c.Parent_Details();
c.Child1_Details();
c.Child2_Details();  } }
```

Hierarchical Inheritance:

In Hierarchical Inheritance, one class is inherited by many sub classes.

```
class Parent
{
void Parent_Details()
{
System.out.println("Parent Details...");
}
}

class Child1 extends Parent
{
void Child1_Details() {
System.out.println("Child1 Detail...");
}
}

class Child2 extends Parent {
void Child2_Details() {
System.out.println("Child2 Detail...");
}
}

public class Hierarichial_Inheritance
{
public static void main(String args[])
{
Child1 c = new Child1();
Child2 c1 = new Child2();
c.Parent_Details();
c.Child1_Details();
c1.Parent_Details();
c1.Child2_Details();} }
```

Hybrid Inheritance:

Hybrid inheritance is one of the inheritance types in Java which is a combination of Hierarchical and Multiple inheritance.

Multiple Inheritance:

Multiple Inheritance is one of the inheritances in Java types where one class extends more than one class. Java does not support multiple inheritance.

Exception

An **unexpected event** that disrupts the normal flow of program execution is called Exception.

Types of Java Exceptions

1) **Checked Exception**
The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions
Checked exceptions are checked at compile-time.
For example, IOException, SQLException, ClassNotFoundException.

2) **Unchecked Exception**
The classes that inherit the RuntimeException are known as unchecked exceptions.
Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
3) **Error**

Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

1. what is exception handling and its advantages?

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

1. **Throwing an exception:** When an exceptional condition occurs, the program throws an exception object containing information about the error.
2. **Catching an exception:** You can use try-catch blocks to catch specific exceptions or general Exception class to catch any exception.

Syntax of Java try-catch

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw an exception
}finally{}
```

Example using try-catch statements

ArithmeticException

```
public class TryCatchExample
{
public static void main(String args[])
{
try
{
//code that may raise exception
int data=100/0;
}
```

```
catch(ArithmeticException e)
{
System.out.println(e);
}
//rest code of the program
System.out.println("rest of the code...");
}
```

NullPointerException

```
public class TryCatchExample1
{
public static void main(String args[])
{
try
{
//code that may raise exception
String s=null;
System.out.println(s.length());
}
catch(NullPointerException e)
{
System.out.println(e);
}
//rest code of the program
System.out.println("rest of the code...");
}
```

NumberFormatException

```
public class TryCatchExample1
{
public static void main(String args[])
{
try
{
//code that may raise exception
String s="abc";
int i=Integer.parseInt(s);
}
catch(NumberFormatException e)
{
System.out.println(e);
}
//rest code of the program
```

```
System.out.println("rest of the code...");
}
```

ArrayIndexOutOfBoundsException

```
public class TryCatchExample1
{
public static void main(String args[])
{
try
{
//code that may raise exception
int a[]=new int[5];
a[10]=50;
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println(e);
}
//rest code of the program
System.out.println("rest of the code...");
}
```

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Notes:

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

```
public class MultipleCatchBlock1 {
public static void main(String[] args) {
try{
int a[]=new int[5];
a[5]=30/0;
```

```
}
catch(ArithmeticException e)
{
System.out.println("Arithmetic Exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("ArrayIndexOutOfBounds Exceptio
n occurs");
}
catch(Exception e)
{
System.out.println("Parent Exception occurs");
}
System.out.println("rest of the code");
}
}
```

Java Nested try block

- In Java, using a try block inside another try block is permitted. It is called a nested try block.
- Every statement that we enter in a statement in try block, the context of that exception is pushed onto the stack.

```
public class NestedTryBlock{
public static void main(String args[]){
//outer try block
try{
//inner try block 1
try{
System.out.println("going to divide by 0");
int b =39/0;
}
//catch block of inner try block 1
catch(ArithmeticException e)
{
System.out.println(e);
}
//inner try block 2
try{
int a[]=new int[5];
```

```
//assigning the value out of array bounds
a[5]=4;
}
//catch block of inner try block 2
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println(e);
}
System.out.println("other statement");
}
//catch block of outer try block
catch(Exception e)
{
System.out.println("handled the exception (outer catch)");
}
System.out.println("normal flow..");
}
}
```

Throw and throws

Both `throw` **and** `throws` keywords are used in Java for exception handling, but they serve distinct purposes:

throw:

- Used **inside a method or block of code** to explicitly **throw an exception object**.
- Takes an **instance of the exception** to be thrown as an argument.
- Used for **signaling errors or unexpected conditions** during runtime.
- Can only throw **one exception at a time**.

throws:

- Used in the **method signature** to **declare the exceptions that the method might throw**.
- Takes a **list of exception classes** as arguments, separated by commas.
- Informs the caller about potential **exceptions that need to be handled**.
- Can declare both **checked and unchecked exceptions**.

Multithreading

Multithreading is a process of executing multiple threads simultaneously.
A thread is a lightweight sub-process, the smallest unit of processing.
Multiprocessing and multithreading, both are used to achieve multitasking.

● Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilise the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

● Thread

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.
Threads are independent. If an exception occurs in one thread, it doesn't affect other threads. It uses a shared memory area.

Life cycle of thread:

1. New:

- When a new thread object is created using the `new Thread()` constructor, it enters the `NEW` state.
- In this state, the thread's code hasn't been allocated any resources or scheduled for execution.

2. Runnable:

- When the `start()` method is called on the thread object, it moves to the `RUNNABLE` state.

- This state indicates that the thread is ready to run, meaning it has the necessary resources and is waiting for its turn to be scheduled for execution by the thread scheduler.

3. Running:

- When the thread scheduler selects the `RUNNABLE` thread, it enters the `RUNNING` state.
- In this state, the thread executes its code on the available processor.
- Only one thread can be in the `RUNNING` state on a single processor at a time.

4. Blocked/Waiting:

- A thread can enter the `BLOCKED` state due to various reasons like:
  - Attempting to acquire a lock that's already held by another thread.
  - Waiting for input from a user or I/O operation.
  - Calling methods like `wait()` or `join()` on other threads.
- In the `WAITING` state, the thread is inactive but still alive, waiting for a specific condition to occur before resuming execution.

5. Timed Waiting:

- Similar to `WAITING`, but the thread waits for a specific duration instead of an indefinite period.
- This state is used when you want to limit the waiting time of a thread to avoid potential deadlocks.

6. Terminated:

- A thread reaches the `TERMINATED` state when:
  - It finishes executing its code successfully.
  - It throws an uncaught exception and doesn't have a `try-catch` block to handle it.
  - It's explicitly stopped using methods like `stop()` or `interrupt()`.
- In this state, the thread is dead and its resources are released.

How to create a thread in Java

There are two ways to create a thread:
1. By extending Thread class
2. By implementing Runnable interface.

• Thread class:

Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interfaces have only one method named run().
public void run(): is used to perform action for a thread.

Collection

Collection

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- A framework is a set of classes and interfaces which provide a ready-made architecture.
- In order to implement a new feature or a class, there is no need to define a framework.
- However, an optimal object-oriented design always includes a framework with a collection of classes such that all the classes perform the same kind of task.
- Java Collections can achieve all the operations that you perform on data such as searching, sorting, insertion, manipulation, and deletion.

● Java Collection means a single unit of objects.

2. What is the difference between `ArrayList` and `LinkedList`?
Answer:

- **`ArrayList`**:
  - Backed by a dynamic array.
  - Provides fast random access
  - Inefficient for insertions and deletions except at the end of the list.
  - More memory efficient for storing and accessing data.

- **`LinkedList`**:
  - Backed by a doubly linked list.
  - Provides sequential access (O(n) time complexity for get operations).
  - Efficient for insertions and deletions (O(1) time complexity) at any position.
  - Requires more memory due to storing two pointers (previous and next).

3. Explain the differences between `HashMap` and `TreeMap`.
Answer:

- **`HashMap`**:
  - Stores entries in a hash table.
  - //Provides O(1) time complexity for get and put operations.
  - Does not maintain any order of keys.
  - Allows one null key and multiple null values.

- **`TreeMap`**:
  - Stores entries in a Red-Black tree
  - Provides O(log n) time complexity for get and put operations.
  - Maintains keys in natural order or a specified comparator order.
  - Does not allow null keys (but allows null values).

4. What is a `Set` in Java Collections Framework and what are its implementations?

Answer: A **`Set`** is a collection that does not allow duplicate elements. It models the mathematical set abstraction.

- **`HashSet`**:
  - Backed by a hash table.
  - Does not maintain any order.
  - Allows one null element.
- **`LinkedHashSet`**:
  - Backed by a hash table and a linked list.
  - Maintains insertion order.
  - Allows one null element.
- **`TreeSet`**:
  - Backed by a **`TreeMap`**.
  - Maintains elements in natural or specified comparator order.
  - Does not allow null elements.

5. What is the difference between `Iterator` and `ListIterator`?

Answer:

- **`Iterator`**:
  - Can traverse elements in a collection in a forward direction only.
  - Provides methods like **`hasNext()`**, **`next()`**, and **`remove()`**.
- **`ListIterator`**:
  - Can traverse elements in both forward and backward directions.
  - Provides additional methods like **`hasPrevious()`**, **`previous()`**, **`add()`**, **`set()`**, in addition to **`hasNext()`**, **`next()`**, and **`remove()`**.

6. How does `ConcurrentHashMap` differ from `HashMap`?

Answer:

- **`HashMap`**:
  - Not synchronized.
  - Suitable for single-threaded environments.
  - Can be externally synchronized using **`Collections.synchronizedMap()`**, but this can lead to contention and reduced performance.

- **`ConcurrentHashMap`**:
    - Designed for concurrent access.
    - Uses a fine-grained locking mechanism (segments or buckets) to allow concurrent read and write operations.
    - Higher throughput in a multi-threaded environment compared to **`HashMap`** wrapped with **`synchronizedMap()`**.

7. What is the purpose of the `Collections` utility class?

Answer: The **`Collections`** utility class provides static methods for operating on collections. These methods include:

- Sorting (**`sort()`**).
- Searching (**`binarySearch()`**).
- Reversing (**`reverse()`**).
- Shuffling (**`shuffle()`**).
- Making collections unmodifiable (**`unmodifiableList()`**, **`unmodifiableSet()`**, **`unmodifiableMap()`**).
- Synchronizing collections (**`synchronizedList()`**, **`synchronizedSet()`**, **`synchronizedMap()`**).

8. What is the difference between `Comparable` and `Comparator`?

Answer:

- **`Comparable`**:
    - Used to define the natural ordering of objects.
    - The class that implements **`Comparable`** must override the **`compareTo()`** method.
    - Can compare an object with itself.

- **`Comparator`**:
    - Used to define an external custom ordering.
    - The class that implements **`Comparator`** must override the **`compare()`** method.
    - Can compare two different objects of the same or different classes.

9. How does `HashSet` handle duplicates?

Answer: **`HashSet`** uses the **`hashCode()`** and **`equals()`** methods to determine whether two objects are the same. When an element is added to a **`HashSet`**, it calculates the hash code of the element and checks if the hash code already exists in the set. If it does, it then uses the **`equals()`** method to check for equality. If both **`hashCode()`** and **`equals()`** indicate that the element is a duplicate, it is not added to the set.

10. What are the main characteristics of `PriorityQueue`?

Answer:

- It is an unbounded queue based on a priority heap.
- Elements are ordered based on their natural ordering or by a comparator provided at queue construction time.
- The head of the queue is the least element with respect to the specified ordering.
- It does not allow `null` elements.
- It provides methods like `offer()`, `peek()`, `poll()`, which work in O(log n) time for the basic operations.

These questions cover a wide range of topics in Java collections and provide a good foundation for understanding and explaining the key concepts and differences between various collection types.

Java Threading Interview Questions and Answers

1. What is a thread in Java?

Answer: A thread in Java is a lightweight process that allows multiple tasks to be executed concurrently within a single program. Each thread runs independently but shares the same memory space. Java provides built-in support for multithreading through the `java.lang.Thread` class and the `java.util.concurrent` package.

2. What are the main states of a thread in Java?

Answer: A thread in Java can be in one of the following states:

- **New**: The thread is created but not yet started.
- **Runnable**: The thread is ready to run and is waiting for CPU time.
- **Blocked**: The thread is waiting for a monitor lock to enter a synchronized block/method.
- **Waiting**: The thread is waiting indefinitely for another thread to perform a particular action.
- **Timed Waiting**: The thread is waiting for another thread to perform a particular action for a specified waiting time.
- **Terminated**: The thread has completed its execution or has been aborted.

3. How do you create a thread in Java?

Answer: There are two main ways to create a thread in Java:

1. **Extending the `Thread` class**:

```
public class MyThread extends Thread {

    public void run() {
```

```
        // Code to be executed in the new thread
    }
}
MyThread thread = new MyThread();
thread.start();  // Starts the thread
```

2.Implementing the `Runnable` interface:

```
public class MyRunnable implements Runnable {
    public void run() {
        // Code to be executed in the new thread
    }
}


Thread thread = new Thread(new MyRunnable());
thread.start();  // Starts the thread
```

4. What is the difference between `Runnable` and `Callable`?

Answer:

- **`Runnable`**:
  - Does not return a result.
  - Cannot throw checked exceptions.
  - Defined with the **`run()`** method.
- **`Callable`**:
  - Returns a result of type **`T`**.
  - Can throw checked exceptions.
  - Defined with the **`call()`** method.

Example of **`Callabl:`**

```
import java.util.concurrent.Callable;



public class MyCallable implements Callable<Integer> {

    public Integer call() throws Exception {

        // Code to be executed in the new thread

        return 123;  // Example result

    }

}
```

5. How do you synchronize a method or a block of code in Java?
Answer: To synchronize a method or block of code, you use the `synchronized` keyword.

- **Synchronized method**:

```
public synchronized void synchronizedMethod() {

    // Code to be synchronized

}public synchronized void synchronizedMethod() {

    // Code to be synchronized

}
```

- **Synchronized block**:

```
public synchronized void synchronizedMethod() {

    // Code to be synchronized

}
```

6. What is the purpose of `wait()`, `notify()`, and `notifyAll()` methods in Java?
Answer: These methods are used for inter-thread communication and must be called within a synchronized context.

- **`wait()`**: Causes the current thread to wait until another thread invokes **`notify()`** or **`notifyAll()`** on the same object.

- **`notify()`**: Wakes up a single thread that is waiting on the object's monitor.
- **`notifyAll()`**: Wakes up all the threads that are waiting on the object's monitor.

Example:

```
synchronized (lock) {
    while (condition) {
        lock.wait();  // Wait for condition to change
    }
    // Perform actions after being notified
    lock.notify();  // Optionally notify other waiting threads
}
```

7. What is a `ThreadPool` and why is it used?

Answer: A **`ThreadPool`** is a pool of worker threads that are reused to execute multiple tasks. It helps in managing a large number of short-lived tasks by reusing a fixed number of threads, thus reducing the overhead of thread creation and destruction.

Java provides **`ThreadPool`** implementations through the **`Executor`** framework:

```
ExecutorService executor = Executors.newFixedThreadPool(10);
executor.submit(new RunnableTask());
executor.shutdown();
```

8. Explain the differences between `synchronized` and `ReentrantLock`.

Answer:

- **`synchronized`**:
  - A keyword in Java that provides a simple syntax for mutual exclusion.
  - Implicit locking and unlocking.
  - Less flexible, no try-lock mechanism, no fairness policy.
- **`ReentrantLock`**:

- A class in `java.util.concurrent.locks` package.
- Explicit locking and unlocking.
- Provides more flexibility with features like try-lock, timed-lock, and fairness policy.
- Example:

```
ReentrantLock lock = new ReentrantLock();
lock.lock();
try {
    // Critical section
} finally {
    lock.unlock();
}
```

9. What is the `volatile` keyword in Java?

Answer: The `volatile` keyword is used to indicate that a variable's value will be modified by different threads. It ensures that changes to the variable are immediately visible to all threads. It also prevents instruction reordering, providing a lightweight synchronization mechanism.

Spring Boot Interview Questions and Answers

1. What is Spring Boot and what are its main features?

Answer: Spring Boot is an extension of the Spring Framework that simplifies the process of building production-ready applications. It provides a range of features that help developers create standalone, production-grade Spring applications quickly and with minimal configuration.

Main features include:

- **Auto-configuration**: Automatically configures Spring and third-party libraries based on the project's classpath.
- **Standalone applications**: Allows the creation of standalone applications with embedded servers (Tomcat, Jetty, etc.).
- **Spring Initializer**: A web-based tool to bootstrap a new project quickly.
- **Production-ready metrics and monitoring**: Includes endpoints to monitor and manage application health and metrics.
- **Convention over configuration**: Reduces the need for extensive configuration by following sensible defaults.