­­­­­­­

**1.** What is java?

* Java is a programming language and a platform. Java is a high-level, robust, object oriented and secured programming language.

**2.** Tell me some features of java­­­?

* **Simple**: Java is very easy to learn, and its syntax is simple, clean and easy to understand.
* **Object-oriented**: Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour.
* **Platform Independent**: Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.
* **Compile and Interpreted**: Java Language offers both compilation and interpretation of programs. when java program is created the java compiler compiles the source code into byte code. the java interpreter converts java byte code into machine code.
* **Secured**: Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
* No explicit pointer
* Java Programs run inside a virtual machine sandbox
* **Robust**: The English mining of Robust is strong. Java is robust because:
* It uses strong memory management.
* There is a lack of pointers that avoids security problems.
* **Portable**: Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
* **High-performance**: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
* **Distributed**: Java is distributed because it facilitates users to create distributed applications in Java.
* **Multi-threaded:** A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

3. What is difference between JDK, JRE, & JVM?

**JVM**:

* JVM is an abstract machine. It is called as virtual machine because it does not physically exist.
* It is a specification that provides runtime environment.

**JRE:**

* JRE is an acronym for java run time environment. It is a set of software tools that is used for developing java applications.
* It is used to provide the run time environment.

**JDK:**

* JDK is an acronym for java development kit.
* The JDK is a software development environment which is used to develop java application & applets. It contains JRE + development tools.

+ Development Tools

**JDK**

**JRE**

+ Libraries

**3.** What is Class Loader in java?

* In Java, class loaders are a fundamental part of the Java Runtime Environment (JRE) responsible for loading classes and resources at runtime.
* The primary purpose of class loaders is to dynamically find and load Java class files into memory when they are needed by the Java program.
* Java class loaders form a hierarchy and work together to load classes as needed. There are three main types of class loaders in Java:

1. Bootstrap Class Loader:
2. Extensions Class Loader:
3. System Class Loader (also known as Application Class Loader):

**4.** How many types of memory areas are allocated by JVM?

* JVM Is the actual one that calls the main method in java application. it is a part of jre.

1. Class (Method) Area
2. Heap
3. Stack
4. Program Counter Register
5. Native Method Stack

**5**. What will be the value of a local variable?

* The local variables do not have any default values in Java.
* This means that they can be declared and assigned a value before the variables are used for the first time, otherwise, the compiler throws an error.
* The local variable i had declared without a value like (int i;), it will compile, but if u try to access it (syso(i)), then it says local variable has not been initialised.

**6**. Describe the process of memory allocation in java?

* In Java, memory allocation primarily refers to the allocation of memory for objects and variables during the runtime of a Java program.
* Java memory allocation is managed by the Java Virtual Machine (JVM), which is responsible for dynamically allocating and deallocating memory as needed.

**Heap Memory**: Objects are stored in the heap memory, shared among all threads.

**Stack Memory**: Local variables are stored in the stack memory, using a last-in-first-out approach.

**Garbage Collection**: Automatic memory management system that reclaims memory no longer referenced by live objects.

**Memory Management**: Techniques such as generational garbage collection and compaction are used for efficient memory management.

**Primitive Types**: Memory allocation for int, float, char, etc., is done directly on the stack and deallocated when out of scope.

**Arrays**: Memory for array elements is allocated on the heap, and elements are stored consecutively in memory.

**7.** Explain difference between Heap Memory and Stack Memory?

**Heap Memory:**

* The heap is a region of memory used for dynamic memory allocation.
* It is where objects and their associated data are stored during the execution of a Java program.

**Stack Memory:**

* The stack is a region of memory used for storing method call frames, local variables, and method call information.
* Each thread in a Java program has its own stack, which stores data related to the methods being executed by that thread.

**8.** Data Types and size of DT?

1. Byte - 1(byte)
2. Short - 2 bytes
3. Int - 4 bytes
4. Long - 8 bytes
5. Float - 4 bytes
6. Double - 8 bytes
7. Char - 2 bytes
8. Boolean - 1 bit

**9. ­­­­**Explain System.out.println?

* main represents the starting point of program.
* string args [] or string [] args are used for command line argument.
* system.out. println is used to print statement. here system is a class, out is an object of the print stream class, println is a method of Print Stream class.

**10.** Order of execution of java programme?

* In Java, the order of execution refers to the sequence in which different parts of a program are executed.

Generally, Java programs are executed in the following order:

1. **Static blocks and variable initialization**: Static variables and static blocks are initialized and executed before the main method is called. This happens only once, when the class is loaded by the JVM.

2. **Main method execution**: The main method is the entry point of any Java program. It is executed after the static blocks and variable initialization.

3. **Instance initialization blocks and variable initialization**: Instance variables and instance initialization blocks are executed when an object of the class is created.

4. **Constructor execution**: Constructors are special methods that are used to initialize objects. When an object is created, the constructor corresponding to that class is executed.

5. **Method execution**: After the object is created, methods can be called and executed.

**11**. What is the purpose of the static keyword in Java?

* The static keyword is used to define a class-level member that belongs to the class itself rather than to an instance of the class.
* The static keyword in Java is mainly used for memory management.
* Less memory consumption since they are allocated memory only once when the class is being loaded.

**12.** Why String is Immutable?

* String with the same content share the same storage in a string pool to minimize the copy of the same variable.
* This is to say, once a string is created, it cannot be changed, if any changes it will lead to creation of new strings.

This design decision was made for several reasons, including:

**String Pooling**: Java maintains a string pool (also known as the string constant pool) to store unique string literals. When you create a string literal, Java checks if an identical string already exists in the pool. If it does, the existing instance is returned instead of creating a new one. This saves memory and improves performance.

**Security**: Strings are often used as parameters in many Java classes, including those representing network connections, database connections, and files. By making strings immutable, the risk of data tampering is reduced.

**Thread Safety**: Immutable objects are inherently thread-safe. Since strings cannot be modified after creation, multiple threads can safely share the same string without synchronization issues.

**Performance**: Immutability makes strings more predictable and allows certain optimizations to be applied by the Java Virtual Machine (JVM), leading to potential performance improvements.

**13.** How Java provide high Performance?

* Java uses Just-In-Time compiler to enable high performance.
* Just-In-Time compiler is a program that turns Java bytecode into instructions that can be sent directly to the processor.

**14.** what is super keyword in java?

* The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor.
* The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.

**15.** what is This keyword is used for in java?

**‘this’ keyword**: ‘this’ keyword in Java that refers to the current instance of the class. In OOPS it is used to:

1. pass the current object as a parameter to another method

2. refer to the current class instance variable

**16.** What are wrapper classes in java?

* Wrapper classes in Java are a set of classes that allow you to convert primitive data types into objects and vice versa.
* Wrapper classes provide a way to treat these primitive types as objects and enable various utility methods to be applied to them.
* Wrapper classes are immutable classes.

The following are the wrapper classes in Java for each primitive data type:

1. **Integer**: Wrapper for int
2. **Long**: Wrapper for long
3. **Short**: Wrapper for short
4. **Byte**: Wrapper for byte
5. **Float**: Wrapper for float
6. **Double**: Wrapper for double
7. **Character**: Wrapper for char
8. **Boolean**: Wrapper for Boolean

**17.** How do you convert primitive data type into wrapper class in java?

* In Java, you can convert a primitive data type into its corresponding wrapper class using a process called "boxing" or "wrapping."

int primitiveInt = 42;

Integer wrappedInt = Integer.valueOf(primitiveInt); // Using valueOf ()

// or

Integer wrappedInt = new Integer(primitiveInt); // Using constructor (deprecated in Java 9+)

* Also, Java provides a feature called "autoboxing," where the Java compiler automatically converts primitive types to their corresponding wrapper classes when necessary, and "unboxing," where the wrapper objects are automatically converted back to their primitive types.

int primitiveInt = 42;

Integer wrappedInt = primitiveInt; // Autoboxing

int unwrappedInt = wrappedInt; // Unboxing

**18.** What is polymorphism?

* Polymorphism is the ability to present the same interface for differing underlying forms (data types).
* Types of Polymorphism IMP

1. Compile Time Polymorphism (Static)

2. Runtime Polymorphism (Dynamic)

* Compile Time Polymorphism: The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Example - Method Overloading
* Method Overloading: Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. The type of the parameters passed to the function.

2. The number of parameters passed to the function.

* Runtime Polymorphism: Runtime polymorphism is also known as dynamic polymorphism. Function overriding is an example of runtime polymorphism.
* Function overriding means when the child class contains the method which is already present in the parent class. Hence, the child class overrides the method of the parent class.
* In case of function overriding, parent and child classes both contain the same function with a different definition. The call to the function is determined at runtime is known as runtime polymorphism.

1. What is loose coupling & tight coupling?

* Tight coupling means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability of the code.
* While Loose coupling means reducing the dependencies of a class that uses the different class directly.

**20.** Why Multiple inheritance is not supported in java?

* To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
* Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. if A and B classes have the same method and you called it from child class object, there will be ambiguity to call the method of A or B.
* Since compile time errors are better than run time errors, java renders compile time error if u inherit 2 classes.

**21**. Difference between Abstraction and Encapsulation?

* Abstraction is the method of hiding the unwanted information.
* Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
* We can implement abstraction using abstract class and interfaces.

**22.** What is Abstraction?

* Abstraction in Java is a process of hiding the implementation details from the user and showing only the functionality to the user.
* In java abstraction is achieved through abstract classes and interfaces.
* An abstract class in Java is a class that cannot be instantiated directly, meaning you cannot create objects of that class.
* Instead, it serves as a blueprint for other classes to extend and implement.
* Abstract classes can contain both abstract methods (methods without a body) and concrete methods (methods with a body).
* Abstract methods must be implemented by the concrete subclasses that extend the abstract class.

abstract class Shape {

// Abstract method - no method body

public abstract double calculateArea ();

// Concrete method

public void display () {

System.out.println("This is a shape.");

}

}

**23**. What is an interface in Java?

* An interface in Java is a mechanism that is used to achieve complete abstraction.
* It is basically a kind of class that contains only constants and abstract methods.

**24**. Difference between interface and abstract class ?

INTERFACE:

1. If we don’t have anything about implementation just, we have requirement specification then we

should go for interface.

2. Inside interface every method is abstract and public. Hence, we achieve 100% abstraction.

3. Interface can have only abstract methods. Since Java 8, it can have default and static methods

also.

4. Interface supports multiple inheritance.

5. Interface has only static and final variables.

6. Interface can't provide the implementation of abstract class.

7. The interface keyword is used to declare interface.

8. An interface can be implemented using keyword "implements".

9. Members of a Java interface are public by default.

ABSTRACT CLASS:

1. If we are talking about implementation but not completely then we should go for abstract class.

2. Every method present in abstract class need to be abstract and public. we can also have concrete

methods also.

3. Abstract class can have abstract and non-abstract methods.

4. Abstract class doesn't support multiple inheritance.

5. Abstract class can have final, non-final, static and non-static variables.

6. Abstract class can provide the implementation of interface.

7. The abstract keyword is used to declare abstract class.

8. An abstract class can be extended using keyword "extends".

9. A Java abstract class can have class members like private, protected, etc.

* Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

**25**. can we override static method in multiple inheritance in java?

* In Interface, these static methods contain the complete definition of the function and since the definition is complete and the method is static, therefore these methods cannot be overridden or changed in the implementation class.

**26**. Can we define private and protected modifiers for data members (fields) in interfaces?

* No, we cannot define private and protected modifiers for variables in interface because the fields (data members) declared in an interface are by default public, static, and final.

**27**. Exception Hierarchy?

* Exception is an event that disturbs the normal flow of the programme. It is an object which is thrown at run time.
* Exception handling is an mechanism that handles the compile & run time errors.

<https://media.geeksforgeeks.org/wp-content/uploads/20220120111809/Group21-660x330.jpg>

**28**. What is the difference between throw and throws in Java?

* The throw keyword is used to explicitly throw an exception in a method or block of code.
* The throws keyword is used in the method signature to indicate that the method may throw one or more exceptions and that the caller of the method needs to handle or propagate those exceptions.

**29**. What are checked and unchecked exceptions in Java?

* Checked exceptions are exceptions that need to be declared in the method signature using the throws keyword or handled using a try-catch block.
* Examples include IO Exception, SQL Exception, ClassNotFoundException, Socket Exception, FileNotFoundException etc.
* Unchecked exceptions, also known as runtime exceptions, do not need to be declared or caught explicitly. They are subclasses of Runtime Exception or Error. Examples include NullPointerException, ArrayIndexOutOfBoundsException, etc.

**30**. What is the purpose of native method in Java?

* In Java, a native method is a method whose implementation is provided in a language other than Java.
* These methods are declared with the native keyword and serve as a bridge between Java code and code written in other programming languages, such as C or C++.

**31.** What is aggregation and composition in java?

* Aggregation and composition are two important concepts in object-oriented programming (OOP) that describe the relationship between two classes.

**Aggregation**:

* Aggregation is a relationship between two classes where one class contains a reference to another class as a member variable.
* An example of aggregation is a university containing multiple departments, where each department can exist independently of the university.

------------------

class Department {

private String name;

private String code;

public Department (String name, String code) {

this.name = name;

this.code = code;

}

}

class University {

private String name;

private String location;

private List<Department> departments; // Aggregation: University has-a list of departments.

public University(String name, String location) {

this.name = name;

this.location = location;

this.departments = new ArrayList<>();

}

public void addDepartment(Department department) {

departments.add(department);

}

}

// Usage:

Department dept1 = new Department("Computer Science", "CS");

Department dept2 = new Department("Physics", "PHYS");

University university = new University("ABC University", "City XYZ");

university.addDepartment(dept1);

university.addDepartment(dept2);

--------------

class Engine {

private int horsepower;

public Engine(int horsepower) {

this.horsepower = horsepower;

}

}

class Car {

private String make;

private String model;

private Engine engine; // Composition: Car has-a Engine.

public Car(String make, String model, int horsepower) {

this.make = make;

this.model = model;

this.engine = new Engine(horsepower);

}

public int getHorsepower() {

return engine.getHorsepower();

}

}

// Usage:

Car car = new Car("Toyota", "Camry", 200);

int horsepower = car.getHorsepower();

----------------

**Composition**:

* Composition is a relationship between two classes where one class contains another class as a member variable, and the contained class cannot exist independently of the containing class.
* When the containing class is destroyed, the contained class is also destroyed.
* An example of composition is a car containing an engine, where the engine cannot exist without the car.

**32**. Explain the transient keyword in Java?

* The transient keyword is used to indicate that a variable should not be serialized when an object is converted to a byte stream (e.g., for storage or network transmission) using Java's Serialization API.

**COLLECTIONS FRAMEWORK**

1. What is the difference between Java collection and Java collections?

* Collection is an interface where you group objects in to single unit.
* Collections is a utility class that has some set of operations you perform on collection.
* Collection doesn’t have all static methods in it, where Collections consist methods that are all static.

1. What is the difference between Collection and Collections?

* The differences between the Collection and Collections are given below.
* The Collection is an interface whereas Collections is a class.
* The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
* The Collection interface provides the standard functionality of data structure to List, Set, and Queue.
* However, Collections class is to sort and synchronize the collection elements.
* The Collection interface provides the methods that can be used for data structure.
* Whereas Collections class provides the static methods which can be used for various operation on a collection.

1. What is Iterator?

* In Java, an iterator is an interface defined in the java. util package that provides a way to iterate over the elements of a collection in a sequential manner.
* hasNext (): returns true if there is at least one more element in the collection, otherwise false.
* next (): returns the next element in the collection.
* remove (): removes the last element returned by the iterator from the collection.

1. Collections Hierarchy?

* <https://data-flair.training/blogs/wp-content/uploads/sites/2/2018/03/hierarchy-of-collection-framework-in-java.webp>

**5.** How HashMap internally works?

* A HashMap uses a hash table, however it internally implemented by 2 data structures namely arrays and linked list.
* Whenever you declare a HashMap, internally it will create an array of buckets. The buckets are referred to as nodes or you can say as list.
* Internally HashMap uses a hash Code of the key Object and this hash Code is further used by the hash function to find the index of the bucket where the new entry can be added.
* HashMap uses multiple buckets and each bucket points to a Singly Linked List where the entries (nodes) are stored.

**6.** What is the difference between Array List and LinkedList?

* Array List is implemented as a resizable array and provides fast random access but slower insertion and deletion of elements in the middle of the list.
* LinkedList is implemented as a doubly-linked list and provides fast insertion and deletion of elements but slower random access.
* Use Array List when you need fast access to elements and LinkedList when you frequently need to add or remove elements from the list.

**7.** Does set allows null and duplicate values?

* In Java, a Set is a collection that does not allow duplicate elements. That means that if you try to add an element to a Set that already exists in the Set, the new element will not be added.
* HashSet and LinkedHashSet both allow null values.
* Tree Set does not allow null values. If you try to add a null value to a Tree Set, it will throw a NullPointerException.

**8.** What is Map in java?

* In Java, a Map is a collection that maps keys to values, where each key can only be associated with a single value.
* Map does not allow duplicate keys, but it does allow null values.
* You can have a null value associated with a key in a Map, but you cannot have two keys with the same value. If you try to put a value into a Map using a key that already exists in the Map, the new value will replace the old value associated with that key.

**9.** What is Concurrent HashMap used for?

* Concurrent HashMap is the Map implementation that allows us to modify the Map while iteration.
* The Concurrent HashMap operations are thread-safe. Concurrent HashMap doesn't allow null for keys and values.
* ConcurrentHashMap is ideal for high-performance, multi-threaded applications.
* If you need to access and update a map from multiple threads, ConcurrentHashMap is the best option.
* It provides all the operations of a HashMap and additionally allows concurrent access for read, write, and update.

**10.** What are new features introduced with Java 8?

* Lambda Expressions, Interface Default and Static Methods, Method Reference, Parameters Name, Optional, Streams, Concurrency.

**11.** Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

* The difference between Hashtable and ConcurrentHashMap, both can be used in the multithreaded environment, but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for a longer duration.
* Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.
* In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration.

**12.** What is the difference between Iterator and List Iterator?

* Iterator traverses the elements in the forward direction only whereas List Iterator traverses the elements into forward and backward direction.

**13.** What is the difference between List and Set?

* The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.
* The List can contain duplicate elements whereas Set includes unique items.
* The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
* The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
* The List interface can allow n number of null values whereas Set interface only allows a single null value.

**14.** What is the difference between HashSet and Tree Set?

* The HashSet and Tree Set, both classes, implement Set interface. The differences between the both are listed below.
* HashSet maintains no order whereas Tree Set maintains ascending order.
* HashSet implemented by hash table whereas Tree Set implemented by a Tree structure.
* HashSet performs faster than Tree Set.
* HashSet is backed by HashMap whereas Tree Set is backed by Tree Map.

**15**. What is the difference between Set and Map?

* Set contains values only whereas Map contains key and values both.
* Set contains unique values whereas Map can contain unique Keys with duplicate values.
* Set holds a single number of null value whereas Map can include a single null key with n number of null values.

**16.** What is the difference between HashSet and HashMap?

* The differences between the HashSet and HashMap are listed below.
* HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
* HashSet implements Set interface whereas HashMap implements the Map interface
* HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
* HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

**17.** What is the difference between HashMap and TreeMap?

* The differences between the HashMap and TreeMap are given below.
* HashMap maintains no order, but TreeMap maintains ascending order.
* HashMap is implemented by hash table whereas TreeMap is implemented by a Tree structure
* HashMap can be sorted by Key or value whereas TreeMap can be sorted by Key.
* HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

**18.** Tell methods of Collection interface?

1. add
2. add all
3. remove
4. remove all
5. remove if
6. retain all
7. size
8. clear
9. contains
10. containsAll
11. toArray
12. isEmpty
13. stream

**19**. what is difference between AL, LL, VECTOR?

* ArrayList provides fast random access, LinkedList is efficient for frequent insertions and deletions, and Vector is synchronized for thread safety (but slower).
* The choice between them depends on the specific requirements and usage patterns of your application.

**20**. What is the difference between Comparable and Comparator?

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1. Comparable provides only one sort of sequence. | 1. The Comparator provides multiple sorts of sequences. |
| 2. It provides one method named compareTo (). | 2. It provides one method named compare (). |
| 3. If we implement the Comparable interface, The actual class is modified. | 3. The actual class is not changed. |

**21.** Call BY Value?

* Call by value means that the value of the argument is copied and passed to the method.
* This means that any changes made to the parameter within the method are not reflected outside of the method.

public class Example {

public static void main (String[] args) {

int x = 5;

modifyValue(x);

System.out.println(x); // Output: 5

}

public static void modifyValue (int value) {

value = 10;

}

}

**22.** Call by Reference?

* Call by reference means that a reference to the memory location of the argument is passed to the method.

public class Example {

public static void main (String [] args) {

int [] arr = {1, 2, 3};

modifyArray(arr);

System.out.println(Arrays.toString(arr)); // Output: [2, 4, 6]

}

public static void modifyArray (int [] array) {

for (int i = 0; i < array. length; i++) {

array[i] \*= 2;

}

}

}

**23.** What are Cookies?

* A cookie is a small piece of information that is persisted between the multiple client requests.
* A cookie has a name, a single value, and optional attributes.

**24.** What is Entry in Java?

* Entry interface enables you to work with a map entry. The entrySet () method declared by the Map interface returns a Set containing the map entries. Each of these set elements is a Map. Entry object.

**25.** What are JDBC steps?

1. import ---> java.sql

2. Register and load driver.

3. Create connection

4. Create Statement

5. Execute Query

6. process the results

7. close.

**26.** what is PreparedStatement and Statement in jdbc?

* In JDBC, both Statement and PreparedStatement are interfaces used to execute SQL queries against a database. However, they have some differences in their functionality and usage.
* Statement is an interface in JDBC that is used to execute a static SQL statement and return the results. Static SQL statements are those whose values are hard-coded at the time of execution.
* PreparedStatement is used for dynamic SQL statements with parameters, while Statement is used for static SQL statements.
* PreparedStatement provides protection against SQL injection attacks, while Statement does not.

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| 1. It is used for accessing your database. Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. | 1. It is used when you want to use SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |
| 2. If you want to run SQL query only once then this interface is preferred over PreparedStatement. | 2. Used to execute dynamic SQL queries. |
| 3. Used to execute normal SQL queries. | 3. PreparedStatement provides protection against SQL injection attacks, while Statement does not. |

**27**. Difference between Execute and ExecuteUpdate and ExecuteQuery?

* **Execute** (): This method is used for all types of SQL Statements and it returns a Boolean value of TRUE or FALSE. If the method returns "True" it returns a ResultSet object. If False it returns int value.
* **ExecuteUpdate** (): This method is used for execution of DML Statements (Insert, Update, Delete) which is return int value, that is count of rows affected.
* **ExecuteQuery** (): This method is used for retrieving data from database and this method returns ResultSet Object and that returns the data according to the query.

**28**. What is Rowset?

* Rowset Interface is basically an extension of JDBC Set and is part of the java.sql.package
* There are 2 types of RowSet Connected and Dis-Connected.
* A RowSet object may make a connection with a data source and maintain that connection throughout its life cycle, in which case it is called a connected rowset.
* Connected Rowset are those which establish connection and retain until the application terminates.
* Disconnected rowset performs or establishes a connection, executes a query such as retrieving data from the database and closes the connection.
* According to the implementation of rowset there are 5 types of implementations.

1. JdbcRowset

2. Cached Rowset

3. WebRowset

4. Filetered rowset

5. Join Rowset

**29.** what is ResultSet?

* The SQL statements that read data from a database query, return the data in a result set.
* The SELECT statement is the standard way to select rows from a database and view them in a result set.

**30.** What is Servlet?

* A servlet is generally a java-based programme that runs on web server and process client requests and generates dynamic web pages, process and respond to HTTP requests.
* When a client sends a HTTP request to the server, the servlet container receives the request and pass it to appropriate servlet.
* The servlet then processes the request and sends the http response back to the client.
* Servlets can handle various types of HTTP requests such as GET, POST, PUT, and DELETE.

**31.** How many types of servlets?

* There are basically 2 types of servlets. HTTP & Generic Type servlets. we can use Constructor to initialise servlet with the help of init () method and destructor method to remove the servlet from the resources using destroy ().
* There are basically two types of servlets based on the interface they implement.

**Generic Servlet**: It is an abstract class that implements the Servlet, Servlet-Config. It provides a generic implementation of the servlet interface.

**HTTP Servlet**: It is an abstract class that extends Generic servlet and provides Http specific functionality. It implements doGet (), doPost (), doPut(), doDelete() and other methods to handle HTTP requests and responses.

* Both servlets are used to process client requests and generate dynamic responses.
* HTTP servlets are most commonly used in creating web applications that work with http requests and responses.
* While Generic servlets are more commonly used in creating custom servlets that work with other protocols and data formats.

**32.** What is Servlet Context and Servlet Config?

* ServletConfig is used for sharing init parameters specific to a servlet.
* Whereas ServletContext is for sharing init parameters within any servlet with in a web application.

**33.** What is flow of servlets?

1. First the server checks if the servlet is requested for the first time.
2. If yes, web container does the following task:
3. Loads the servlet class.
4. Instantiates the servlet class.
5. Calls the init method passing the servlet config object.
6. Else calls the service method passing request and response object.
7. The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

**34.** What are the 3 main methods of a servlet class?

* The init, service and destroy are the life cycle methods of servlet.

**35.** What are the life-cycle methods for a servlet?

* public void init (ServletConfig config): It is invoked only once when first request comes for the servlet. It is used to initialize the servlet.
* public void service (ServletRequest request, ServletResponse) throws ServletException, IOException : It is invoked at each request. The service() method is used to service the request.
* public void destroy() : It is invoked only once when servlet is unloaded.

**36.** What are phases of servlet life cycle?

1. **Loading a Servlet**: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container.

2. **Initializing the Servlet**: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter.

3. **Handling Request**: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request:

* It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
* After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

4. **Destroying the Servlet**: When a Servlet container decides to destroy the Servlet, it performs the following operations:

* It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
* After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.
* init () and destroy () methods are called only once during the servlet's lifetime, typically during servlet initialization and destruction by the container.
* The service () method is called for each client request, and request-specific handling methods (eg., doGet (), doPost ()) are called as needed based on the request's HTTP method.

**37.­­** Who is responsible to create the object of servlet?

* The web container or servlet container.

**38.** Explain about 4 types of Functional Interfaces in java 8?

* In Java 8, a functional interface is an interface that has only one abstract method. This concept is important in the context of lambda expressions, which are a new language feature introduced in Java 8 for writing concise and expressive code.

Here are the four types of functional interfaces in Java 8:

**Consumer**: The Consumer interface represents an operation that takes a single argument and returns no result. Its functional method is called accept ().

**Supplier**: The Supplier interface represents an operation that takes no arguments and returns a value. Its functional method is called get ().

**Predicate**: The Predicate interface represents a Boolean-valued function of one argument. Its functional method is called test ().

**Function**: The Function interface represents a function that takes one argument and returns a value. Its functional method is called apply ().

**39.** What is the reason for introducing functional interface after java 8?

* Functional Interfaces introduced in Java 8 allow us to use a lambda expression to initiate the interface's method and avoid using lengthy codes for the anonymous class implementation.
* Various built-in interfaces were declared with @FunctionalInterface annotation and made functional from Java 8.

**40.** Why functional interface in java has one abstract method?

* The functional interface also known as Single Abstract Method Interface was introduced to facilitate Lambda functions.
* Since a lambda function can only provide the implementation for 1 method it is mandatory for the functional interface to have only one abstract method.

**41.** What is Stream?

* Stream represents a sequence of objects from a source, which supports aggregate operations.

**42.** What is Completable Future in java?

* CompletableFuture is a class introduced in Java 8 that allows us to write asynchronous, non-blocking code.
* It is a powerful tool that can help us write code that is more efficient and responsive.

**43.** How to implement an immutable class?

* We can make a class immutable by:

1. Making all methods and variables as private.
2. Setting variables within constructor.

Public Class ImmutableClass {

private int member;

ImmutableClass(int var){

member=var;

}

}

and then we can initialize the object of the class as

|  |
| --- |
| ImmutableClass immutableObject = new ImmutableClass(5); |

Now all members being private, you can't change the state of the object.

**44.** What is meant by constructor chaining?

* Constructor chaining is the process of calling a sequence of constructors.
* We can do it in two ways: by using this () keyword for chaining constructors in the same class.
* By using super () keyword for chaining constructors from the parent class.

**45.** What is Enumeration?

* An enumeration (Enum for short) in Java is a special data type which contains a set of predefined constants.

**46.** what is flat map?

* Java flatMap is a method that, when applied to a stream of values, maps each value to some required output value.

**47.** What is the difference between == and. equals() for comparing objects in Java?

* The == operator compares object references, checking if they refer to the same memory location.
* The .equals() method compares the actual content of the objects (the values they hold) based on the implementation of the method in the object's class.
* In other words, == checks for reference equality, while .equals() checks for value equality.

**48.** What is the difference between final, finally, and finalize in Java?

* Final is a keyword used to make a variable, method, or class constant and unchangeable.
* Finally, is a block that follows a try-catch block and is used to ensure that a section of code is always executed, regardless of whether an exception occurs or not.
* Finalize is a method called by the garbage collector before reclaiming the memory occupied by an object. However, it's generally recommended to use the try-with-resources statement or explicitly close resources instead of relying on finalize

**49.** When finally block won’t be executed?

* In Java, there is one possibility where finally block will not be executed. They are as follows:

1. When the System.exit() method is called in the try block before the execution of finally block, finally block will not be executed.

package finallyProgram;

public class FinallyBlock1

{

void m1()

{

try {

System.out.println("I am in try block");

System.exit(0);

}

finally {

System.out.println("I am in finally block");

}

System.out.println("Statement after finally block");

}

public static void main(String[] args)

{

FinallyBlock1 obj = new FinallyBlock1();

obj.m1();

}

}

Output:

I am in try block

---------------

Program code 2:

package finallyProgram;

public class FinallyBlock2 {

void m1()

{

int a = 20, b = 0;

try {

System.out.println("I am in try block");

System.exit(0);

int c = a/b;

System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

System.out.println("I am in catch block");

}

finally {

System.out.println("I am in finally block");

}

System.out.println("Statement after finally block");

}

public static void main(String[] args)

{

FinallyBlock2 obj = new FinallyBlock2();

obj.m1();

}

}

Output:

I am in try block

---------------

Program code 3:

package finallyProgram;

public class FinallyBlock3 {

void m1()

{

int a = 20, b = 0;

try {

System.out.println("I am in try block");

int c = a/b;

System.exit(0);

System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

System.out.println("I am in catch block");

}

finally

{

System.out.println("I am in finally block");

}

System.out.println("Statement after finally block");

}

public static void main(String[] args)

{

FinallyBlock3 obj = new FinallyBlock3();

obj.m1();

}

}

Output:

I am in try block

I am in catch block

I am in finally block

Statement after finally block

**50**. What is try with resources in java?

* The try-with-resources statement is a try statement that declares one or more resources.
* A resource is an object that must be closed after the program is finished with it.
* The try-with-resources statement ensures that each resource is closed at the end of the statement

static String readFirstLineFromFile(String path) throws IOException {

try (FileReader fr = new FileReader(path);

BufferedReader br = new BufferedReader(fr)) {

return br.readLine();

}

}

Here are some examples of commonly used resources that need to be closed after use:

1. **File streams**: When reading from or writing to files.
2. **Database connections:** When interacting with databases.
3. **Network sockets**: When establishing connections to remote servers.
4. **Input/output streams:** When performing input/output operations, such as reading or writing data.
5. **Readers and writers**: Used for reading and writing character data.
6. **JDBC resources**: PreparedStatement, ResultSet, and other JDBC-related objects.
7. **Image resources**: When working with images, such as loading and manipulating them.
8. **Audio resources:** When playing or manipulating audio files.
9. **ZIP file streams**: When reading from or writing to ZIP archives.
10. **BufferedReader and BufferedWriter**: Used for efficient reading and writing of character data.

**51.** Buffer Reader and Buffer Writer?

**Buffer reader**:

* A BufferedReader reads text from a character stream with buffering capabilities, which helps in reducing the number of I/O operations.
* It provides methods like readLine() to read a line of text, read() to read a single character, and more.

try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

// Process each line of text

System.out.println(line);

}

} catch (IOException e) {

// Exception handling

}

**Buffer writer**:

* A BufferedWriter writes text to a character stream with buffering capabilities.
* Similar to BufferedReader, it writes data to an internal buffer and performs fewer I/O operations, which can improve performance.
* It provides methods like write(String) to write a string, newLine() to write a platform-specific line separator, and more.

try (BufferedWriter writer = new BufferedWriter(new FileWriter("file.txt"))) {

writer.write("Hello, world!");

writer.newLine();

writer.write("This is a new line.");

} catch (IOException e) {

// Exception handling

}

**52.** What is the difference between String, StringBuilder, and StringBuffer in Java?

* String is **immutable**, meaning its value cannot be changed once created. Each time you modify a String, a new object is created in memory.
* StringBuilder and StringBuffer are mutable classes used to manipulate strings. They provide methods to append, delete, or modify strings. StringBuilder is not thread-safe, while StringBuffer is thread-safe.
* If you need a mutable string and don't require thread safety, use StringBuilder. If you need thread safety, use StringBuffer.
* StringBuffer is thread-safe and synchronized whereas StringBuilder is not. That's why StringBuilder is faster than StringBuffer.

**53.** List some string builder methods?

1. charAt
2. setCharAt
3. insert
4. deleteCharAt
5. append
6. reverse
7. length

**54**. What are intermediate and terminal operations in java 8?

* In Java 8 and later versions, the Stream API introduced two types of operations: intermediate operations and terminal operations. These operations are used to perform various transformations and computations on a stream of elements.

**Intermediate Operations**:

* Intermediate operations are operations that are applied to a stream and produce a new stream as a result. They are lazy, meaning they do not execute immediately. Instead, they are stored and executed only when a terminal operation is called.

Some common intermediate operations in Java 8 include:

1. **filter**(Predicate<T> predicate): Returns a stream consisting of the elements that match the given predicate.
2. **map**(Function<T, R> mapper): Applies the given function to each element of the stream and returns a stream of the results.
3. **distinct**(): Returns a stream with distinct elements (removes duplicates).
4. **sorted**(): Returns a stream with sorted elements based on their natural order.
5. **limit**(long maxSize): Returns a stream truncated to the specified size.

* Intermediate operations allow you to chain multiple operations together to form a pipeline.

**Terminal Operations**:

* Terminal operations are operations that are applied to a stream and produce a non-stream result, such as a value, a collection, or a side effect. When a terminal operation is called, all the previous intermediate operations are executed.

Some common terminal operations in Java 8 include:

1. **forEach**(Consumer<T> action): Performs an action for each element in the stream.
2. **collect**(Collector<T, A, R> collector): Collects the elements of the stream into a collection or a single value.
3. **count**(): Returns the count of elements in the stream.
4. **anyMatch**(Predicate<T> predicate): Returns true if any elements of the stream match the given predicate.
5. **allMatch**(Predicate<T> predicate): Returns true if all elements of the stream match the given predicate.
6. **findFirst**(): Returns the first element of the stream.

**55.** What are parallel streams in java8?

import java.util.Arrays;

import java.util.List;

public class ParallelStreamExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

// Sequential stream

int sumSequential = numbers.stream()

.filter(n -> n % 2 == 0)

.mapToInt(n -> n)

.sum();

// Parallel stream

int sumParallel = numbers.parallelStream()

.filter(n -> n % 2 == 0)

.mapToInt(n -> n)

.sum();

System.out.println("Sum (Sequential): " + sumSequential);

System.out.println("Sum (Parallel): " + sumParallel);

}

}

* In the example above, we have a list of numbers, and we want to find the sum of the even numbers. The first part demonstrates the sequential stream, where operations are executed sequentially on a single thread.
* The second part shows the parallel stream, where operations are divided among multiple threads for concurrent execution.
* By using parallel streams, Java automatically distributes the workload across available threads and combines the results at the end.
* This can significantly improve the performance of computationally intensive or data-intensive operations when working with large datasets
* However, it's important to note that not all operations are suitable for parallel execution. Some operations, like forEach, rely on the order of elements and may produce incorrect results when executed in parallel.

**56.** What is log4j?

* log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License.
* log4j is a popular logging package written in Java.

**57.** Class methods?

* Object () - constructor

1. getClass()
2. finalize (), its deprecated since 9
3. notify ()
4. notifyAll ()
5. clone ()
6. wait () --> overloaded 3 methods
7. toString()
8. equals ()
9. hashcode()

**58**. What is marker interface in java?

* A marker interface is an interface that doesn't have any methods or constants inside it.
* It provides run-time type information about objects, so the compiler and JVM have additional information about the object.
* Examples of marker interface are Serializable, Cloneable and Remote interface.

**59.** Why do we use Design Patterns?

* Design patterns are a set of proven solutions to common software design problems.
* They provide standardized approaches for designing software systems that are reusable, maintainable, and scalable.
* Java, being an object-oriented programming language, is well-suited for implementing design patterns.

**60**. Design Patterns in java ?

Certainly! There are several design patterns commonly used in Java. Here are some of the most well-known design patterns in Java:

Creational Patterns:

* Singleton: Ensures that a class has only one instance and provides a global point of access to it.
* Factory Method: Defines an interface for creating objects, but lets subclasses decide which class to instantiate.
* Abstract Factory: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
* Builder: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
* Prototype: Creates new objects by cloning existing ones, avoiding the need for complex initialization.

Structural Patterns:

* Adapter: Allows objects with incompatible interfaces to work together by converting the interface of one class into another.
* Decorator: Dynamically adds new behaviours to an object without changing its existing code.
* Composite: Represents a group of objects as a single object, treating individual and composite objects uniformly.
* Proxy: Provides a surrogate or placeholder for another object to control access to it.
* Facade: Provides a simplified interface to a complex subsystem, making it easier to use.

Behavioural Patterns:

* Observer: Defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.
* Strategy: Encapsulates a family of algorithms and makes them interchangeable, allowing clients to switch between algorithms at runtime.
* Command: Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
* Template Method: Defines the skeleton of an algorithm in a base class, allowing subclasses to provide specific implementations of certain steps.
* Iterator: Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

**MULTI-THREADING**

**1.** Explain aboutThreads in java?

* In java, a thread is a unit of execution that allows a programme to perform multiple tasks concurrently.
* Java provides a built-in support for multi-threading the "java.lang.Thread" class.
* A thread is created by instantiating an object of the "Thread" class and calling its method start ().
* When the start () method is called, a new thread is created and its run() method is invoked in separated flow of execution.
* Multithreading is commonly used in Java for tasks that require parallel processing, such as running multiple background tasks, performing I/O operations, and handling user interface events.

// Way 1

// Creating thread By Extending To Thread class

class MyThread extends Thread {

// Method 1

// Run() method for our thread

public void run()

{

// Print statement

System.out.println(

"Thread is running created by extending to parent Thread class");

}

// Method 2

// Main driver method

public static void main(String[] args)

{

// Creating object of our thread class inside main()

// method

MyThread myThread = new MyThread();

// Starting the thread

myThread.start();

}

}

**2.** What is the difference between extending Thread and implementing Runnable for thread creation?

* When extending Thread, you cannot extend any other class, while you can implement multiple interfaces when implementing Runnable.
* Therefore, it's often recommended to implement Runnable to have better design flexibility.

**3.** What are stages of life cycle of a thread in Java?

* Lifecycle and States of a Thread in Java

1. New State
2. Runnable State
3. Blocked State
4. Waiting state
5. Time waiting state
6. Terminate state

4. What is thread pool in java?

* A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.

**5**. What is executor service in java?

* The Java ExecutorService is the interface which allows us to execute tasks on threads asynchronously.
* The Java ExecutorService interface is present in the java. util. concurrent package.
* The ExecutorService helps in maintaining a pool of threads and assigns them tasks.

**6.** What is thread safe in java?

* Thread safety in java is the process to make our program safe to use in multithreaded environment there are different ways through which we can make our program thread safe. Synchronization is the easiest and most widely used tool for thread safety in java.

**7.** What is thread communication, and how is it achieved in Java?

* Thread communication is a way for threads to coordinate and exchange information.
* In Java, you can achieve thread communication using methods like wait(), notify(), and notifyAll() that are provided by the Object class.

**8.** Explain the difference between wait () and sleep () methods in Java.

* wait() is used for thread synchronization and releases the monitor lock, while sleep() is used for pausing a thread's execution without releasing the lock.

**9.** What is thread scheduler?

* A component of Java that decides which thread to run or execute and which thread to wait is called a thread scheduler in Java.
* In Java, a thread is only chosen by a thread scheduler if it is in the runnable state.

**10.** What is thread priority?

* In Java, a thread's priority is an integer in the range 1 to 10. The larger the integer, the higher the priority.
* The thread scheduler uses this integer from each thread to determine which one should be allowed to execute.

**11.** What is synchronization in Java, and why is it important?

* Synchronization in Java is a mechanism to control access to shared resources to ensure thread safety.
* It prevents multiple threads from concurrently accessing or modifying shared data, preventing data corruption and race conditions.

|  |
| --- |
| What is serialistion in java ?  ➳ serialization is the conversion of a Java object into a static stream (sequence) of bytes, which we can then save to a database or transfer over a network.  ➳ serialisation is a process that converting an object's state in a format that can be easily sorted, transmitted, or reconstructed later.  ➳ In java, serialisation is commonly used to persist object to files, send objects over a network, storing objects in db.  This should not be here |

**12.** What is dead lock?

* Dead lock in java is a situation that occur when thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by th first thread. since both the threads are waiting for each other to release the lock simultaneously this condition is called dead lock.

**13.** What is Diff b/w notify and notifyall ?

* In the case of the multithreading, notify () method sends the notification to only one thread among the multiple waiting threads which are waiting for the send lock.
* While notifyAll () methods in the same context send notifications to all waiting threads instead of a single thread.