Java Questions

with Answers to Crack the

Technical Interview

Core Java Concepts:

Q.1 What is the difference between JDK and JRE?

JDK (Java Development Kit) and **JRE (Java Runtime Environment)** are both essential components of the Java platform, but they serve different purposes:

1. **JDK (Java Development Kit):**
   - **Purpose:** JDK is primarily used for Java software development. It contains tools, executables, and binaries needed for writing, compiling, debugging, and running Java applications and applets.
   - **Components:**
     - **Compiler:** The Java Compiler (javac) is part of JDK, which converts Java source code (.java files) into bytecode (.class files).
     - **Debugger:** JDK includes debugging tools for troubleshooting and fixing issues in Java code.
     - **Javadoc:** It provides tools to generate documentation from Java source code comments.
     - **JavaFX:** JDK includes JavaFX, a platform for creating rich internet applications.
     - **Development Libraries:** JDK contains the Java class libraries and API used for Java application development.
   - **Usage:** Developers use JDK to create and compile Java programs. It's required for software development.
2. **JRE (Java Runtime Environment):**
   - **Purpose:** JRE is used to run Java applications and applets. It provides the necessary runtime environment for executing Java code on a computer.
   - **Components:**
     - **Java Virtual Machine (JVM):** JRE includes the JVM, which interprets and executes Java bytecode.
     - **Core Libraries:** It contains the standard class libraries and packages needed for running Java applications.
   - **Usage:** End-users who want to run Java applications need JRE installed on their machines. They don't need JDK unless they are developers.

Q.2 Why is Java a platform independent language?

Java is platform-independent due to its "Write Once, Run Anywhere" principle. It achieves this through bytecode compilation, where Java source code is compiled into bytecode that can run on any platform with a Java Virtual Machine (JVM). This abstraction shields the code from platform-specific details, ensuring portability and consistent execution across different operating systems.

Q.3 What is the difference between an abstract class and an interface?

An abstract class in Java is a class that can have both abstract (unimplemented) and concrete (implemented) methods. It can also have instance variables. It allows for code reuse through inheritance, and a class can extend only one abstract class.

An interface, on the other hand, is like a blueprint for a class. It defines a contract that implementing classes must adhere to. Interfaces can only have abstract methods (public and abstract by default) and constant variables (public, static, and final). A class can implement multiple interfaces.

In summary, the key differences are:

1. **Abstract Class:**
    - **Can have both abstract and concrete methods.**
    - **Can have instance variables.**
    - **Supports single inheritance (a class can extend only one abstract class).**
2. **Interface:**
    - **Can only have abstract methods (no method implementations).**
    - **Can only have constant variables (public, static, final).**

- **Supports multiple inheritance (a class can implement multiple interfaces).**

Q.4 What is the difference between final, finally, and finalize?

In Java, "final," "finally," and "finalize" are three distinct keywords with different meanings and purposes:

final:

- **final is a modifier that can be applied to classes, methods, and variables.**
- **When applied to a class, it indicates that the class cannot be subclassed (i.e., it cannot have child classes).**
- **When applied to a method, it indicates that the method cannot be overridden by subclasses.**
- **When applied to a variable, it means the variable is a constant and cannot be reassigned after initialization.**

finally:

- **finally is a block used in exception handling. It follows the try block and precedes any catch or catch blocks.**
- **The code inside a finally block always executes, whether an exception is thrown or not.**
- **It's typically used for cleanup operations, like closing files or releasing resources, to ensure they are executed regardless of exceptions.**

`finalize`:

- `finalize` is a method that was used for object cleanup in older versions of Java (prior to Java 9).
- When an object becomes unreachable (no longer referenced by any part of the program), the Java Virtual Machine (JVM) calls the `finalize` method before reclaiming memory through garbage collection.
- Starting from Java 9, the use of `finalize` is discouraged, and it has been deprecated. Modern Java encourages using other resource management

techniques, like try-with-resources and explicit resource management, for cleanup operations.

Q.5 What is the difference between stack and heap memory?

In short, stack memory is used for storing method call frames and local variables, providing fast access and automatic memory management. Heap memory, on the other hand, is used for dynamic memory allocation, accommodating objects with varying lifetimes, and requires manual memory management.

Q.6 What is the difference between method overloading and method overriding?

Method overloading, also known as static polymorphism, involves defining multiple methods in the same class with the same name but different parameters. Method overriding, also known as runtime polymorphism, occurs when a subclass provides a specific implementation for a method already defined in its superclass, with the same name, return type, and parameters.

Q.7 What is the difference between an abstract class and an interface?

An abstract class can have both abstract and concrete methods, while an interface can only have abstract methods. A class can extend only one abstract class, but it can implement multiple interfaces.

Q.8 What is the difference between a private and a protected modifier?

The `private` and `protected` modifiers in Java are used to control the access level of class members (fields, methods, and inner classes). Here are the key differences between them:

Private Modifier:

1. Members declared as **private** are the most restricted in terms of access.
2. A **private** member is only accessible within the same class where it is declared. It is not visible to any other class, even subclasses.
3. It provides the highest level of encapsulation, ensuring that the member's state is not directly exposed to external classes.

Protected Modifier:

1. Members declared as protected have a more relaxed access level compared to private. They are accessible within the same class, subclasses, and classes within the same package.
2. A protected member can be inherited by subclasses, even if they are in a different package. It promotes a level of visibility and access suitable for inheritance.

Q.9 What is constructor overloading in Java?

Constructor overloading in Java refers to the practice of defining multiple constructors within a class, each with a different set of parameters. This allows objects of the class to be initialized in various ways, providing flexibility and versatility in object creation. Constructor overloading enables developers to create objects with different initial states or configurations based on the specific needs of the program.

Q.10 What is the use of super keyword in Java?

The super keyword is used to access data members of the parent class when the data members names of the parent class and its child subclasses are the same, to call the default and parameterized constructor of the parent class inside the child subclass and to access parent class methods when the child subclasses have overridden them.

Q.11 What is the difference between static methods, static variables, and static classes in Java?

Static methods, static variables, and static classes in Java all involve the use of the `static` keyword, but they serve different purposes and have distinct characteristics:

Static Methods:

- **Static methods are associated with the class itself rather than with instances (objects) of the class.**
- **They can be called using the class name without creating an instance of the class.**
- **Static methods cannot access instance-specific (non-static) members directly, as they don't have access to this.**

Static Variables (Static Fields):

- **Static variables are also associated with the class itself rather than with instances.**
- **They are shared among all instances (objects) of the class.**
- **Changes to a static variable made by one instance are reflected in all other instances.**
- **Static variables are commonly used for constants or for maintaining values shared across objects.**

Static Classes (Static Nested Classes):

- A static class is a nested class that is marked as `static`.
- It can be accessed using the enclosing class's name, and you don't need to create an instance of the enclosing class.
- Static nested classes are often used for organization and encapsulation of related utility classes.

Q.12 What exactly is System.out.println in Java?

System.out.println() is a method to print a message on the console. System - It is a class present in java.lang package. Out is the static variable of type PrintStream class present in the System class. println() is the method present in the PrintStream class.

Q.13 What part of memory - Stack or Heap - is cleaned in the garbage collection process?

In the garbage collection process in Java, the heap memory is cleaned. The garbage collector's primary role is to identify and reclaim memory occupied by objects that are no longer reachable or in use by the program. These unreferenced objects are typically allocated memory on the heap, so the garbage collector focuses on cleaning up the heap to free up memory for future object allocations. The stack memory, on the other hand, is used for managing method calls and local variables and does not typically involve the garbage collection process.

2. Object-Oriented Programming:

Q.1 What are the Object Oriented Features supported by Java?

Java is an object-oriented programming language and supports the following object oriented features:

- Encapsulation in Java is the fundamental Object-Oriented Programming (OOP) concept that combines data (attributes) and methods (functions) into a single unit called a class. It restricts direct access to an object's data, known as data hiding, and provides public methods (getters and setters) to manipulate and access that data. This helps maintain data integrity, enhances security, and allows for better control over how data is modified and accessed, making code more organized and maintainable. It is one of the key principles of OOP and is essential for building robust and maintainable Java applications.

- Inheritance in Java is a core Object-Oriented Programming (OOP) concept that allows a class (called a subclass or child class) to inherit attributes and methods from another class (called a superclass or parent class). It promotes code reusability and establishes an "is-a" relationship between classes, where a subclass is a specialized version of the superclass. Subclasses can access and extend the behavior of the superclass, enhancing code modularity and facilitating the creation of hierarchical class structures in Java.

- Polymorphism in Java is a fundamental OOP concept that allows objects of different classes to be treated as objects of a common superclass. It enables methods to be defined in a generic way in the superclass and then overridden in the subclasses to provide specialized implementations. Polymorphism promotes code flexibility and reusability by allowing different objects to respond to the same method call differently, based on their specific class implementations. This enhances code modularity and simplifies complex software design.

- Abstraction in Java is a concept that allows you to hide complex implementation details and show only the necessary features of an object. It's achieved through abstract classes and interfaces, enabling you to define a blueprint with method signatures that concrete classes must implement. Abstraction simplifies code maintenance, promotes code reusability, and helps manage complexity by focusing on essential aspects while concealing the underlying complexities of a class or object.

- In Java, classes are blueprint templates for creating objects. They define the structure and behavior of objects by specifying fields (attributes) and methods (functions). Objects, on the other hand, are instances of classes. They represent real-world entities and can interact with each other through method calls, allowing data encapsulation and code reuse in object-oriented programming.

Q.2 What are the different access specifiers used in Java?

Java has 4 access specifiers.

- Public Can be accessed by any class or method.
- Protected Can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.
- the default access modifier is also known as package-private or package-local access. It means that members (fields, methods, and inner classes) with default access are accessible within the same package but not outside of it. They are not visible to classes in other packages. This access modifier is used when no access modifier (public, private, protected) is explicitly specified for a class member.
- Private Can be accessed only within the class.

Q.3 What is the difference between composition and inheritance?

Composition is a "has-a" relationship, where a class contains an object of another class as a member variable. Inheritance is an "is-a"relationship, where a subclass extends a superclass to inherit its attributes and methods.

Q.4 What is the purpose of an abstract class?

An abstract class is a class that cannot be instantiated and is used as a base class for other classes to inherit from. It can contain abstract methods, which are declared but not implemented in the abstract class and must be implemented in the subclasses.

Q.5 What are the differences between constructor and method of a class in Java?

Constructor is used for initialising the object state whereas method is used for exposing the object's behaviour. Constructors have no return type but Methods should have a return type. Even if it does not return anything, the return type is void. If the constructor is not defined, then a default constructor is provided by the java compiler. The constructor name should be equal to the class name. A constructor cannot be marked as final because whenever a class is inherited, the constructors are not inherited. A method can be defined as final but it cannot be overridden in its subclasses.

Q.6 What is the diamond problem in Java and how is it solved?

The diamond problem in Java occurs in multiple inheritance when a class inherits from two classes that have a common ancestor. This creates ambiguity when trying to access methods or fields from the common ancestor through the derived class.

Java solves the diamond problem by allowing multiple interface inheritance, where a class can implement multiple interfaces with the same method names. The implementing class must provide its own implementation of these methods, resolving the ambiguity. This is in contrast to multiple class inheritance, which Java does not support to avoid the diamond problem in the first place.

Q.7 What is the difference between local and instance variables in Java?

Instance variables are accessible by all the methods in the class. They are declared outside the methods and inside the class. These variables describe the properties of an object and remain bound to it. Local variables are those variables present within a block, function, or constructor and can be accessed only inside them. The utilisation of the variable is restricted to the block scope.

Q.8 What is a Marker interface in Java?

A Marker interface in Java is a special type of interface with no method declarations. It serves as a marker or flag to indicate that classes implementing it possess certain characteristics or should be treated differently during runtime.

Common examples of marker interfaces in Java include Serializable and Cloneable. Serializable indicates that objects of the class can be serialized (converted to a byte stream), while Cloneable indicates that objects can be cloned (duplicated) using the clone() method. These marker interfaces enable Java runtime to perform specific operations on objects of classes that implement them.

3. *Data Strctres and Algorithms:*

Q.1 Why are strings immutable in Java?

Strings are immutable in Java to ensure their security, thread-safety, and efficient memory usage. This immutability means once a string is created, its content cannot be changed. Instead, any operation that appears to modify a string actually creates a new string.

1. **Security:** Immutable strings prevent tampering, which is crucial in scenarios like security tokens or passwords.
2. **Thread-Safety:** Multiple threads can read strings simultaneously without the risk of concurrent modification.
3. **Caching:** Immutable strings allow caching, enhancing performance as identical strings can be shared.
4. **Optimization:** Strings can be optimized by the compiler or JVM for better performance.

Q.2 What is the difference between creating a String using new() and as a literal?

If we create a String using new(), then a new object is created in the heap memory even if that value is already present in the heap memory. If we create a String using String literal and its value already exists in the string pool, then that String variable also points to that same value in the String pool without the creation of a new String with that value.

Q.3 What is the Collections Framework?

The Collections Framework in Java is a comprehensive and standardized architecture for representing and manipulating collections of objects. It includes interfaces, classes, and algorithms to manage and organize groups of objects efficiently. The key components of the Collections Framework include:

1. **Interfaces:** These define the abstract data types for different types of collections, such as List, Set, and Map.
2. **Classes:** Java provides concrete implementations of these interfaces, like ArrayList, HashSet, and HashMap.

3. **Algorithms:** The framework offers various algorithms for searching, sorting, and manipulating collections.
4. **Iterators:** These allow sequential access to elements within a collection.

Q.4 What is the difference between ArrayList and LinkedList?

ArrayList and LinkedList are both implementations of the List interface in Java, but they have different characteristics:

1. **Data Structure:**
   - **ArrayList:** It uses a dynamic array to store elements, allowing for fast random access but slower insertions and deletions.
   - **LinkedList:** It uses a doubly-linked list, which enables fast insertions and deletions but slower random access.
2. **Access Time:**
   - **ArrayList:** Accessing elements by index is fast (O(1)), making it suitable for scenarios where random access is frequent.
   - **LinkedList:** Accessing elements by index requires traversing the list from the beginning or end, resulting in slower access times (O(n)).
3. **Insertions and Deletions:**
   - **ArrayList:** Insertions and deletions in the middle of the list are slower (O(n)) due to elements shifting.
   - **LinkedList:** Insertions and deletions at any position are faster (O(1)), as it only involves updating pointers.
4. **Memory Usage:**
   - **ArrayList:** Typically consumes less memory than LinkedList because it doesn't need to store as many pointers.

Q.5 What is the difference between a HashMap and a TreeMap?

`HashMap` and `TreeMap` are both implementations of the `Map` interface in Java, but they have significant differences:

1. **Data Structure:**
   - **HashMap:** It uses a hash table to store key-value pairs. Keys are hashed, allowing for fast retrieval but no specific order.
   - **TreeMap:** It uses a Red-Black tree to store key-value pairs. Keys are sorted in natural order or according to a custom comparator.
2. **Ordering:**
   - **HashMap:** Does not guarantee any specific order of elements. Elements are stored based on their hash codes.
   - **TreeMap:** Maintains elements in sorted order, which can be useful when you need keys sorted in a specific way.
3. **Performance:**
   - **HashMap:** Offers constant-time (O(1)) average-case complexity for basic operations like `get` and `put`.
   - **TreeMap:** Provides guaranteed logarithmic-time (O(log n)) complexity for basic operations because of the tree structure.
4. **Null Keys:**
   - **HashMap:** Allows one `null` key.
   - **TreeMap:** Does not allow `null` keys.
5. **Iterating:**
   - **HashMap:** The order of elements during iteration is not guaranteed and may vary.
   - **TreeMap:** Iterates over elements in sorted order.

Q.6 What is the difference between a HashSet and a TreeSet?

`HashSet` and `TreeSet` are both implementations of the `Set` interface in Java, but they have distinct characteristics:

1. **Ordering:**
   - **HashSet:** Does not maintain any particular order of elements. Elements are stored based on their hash codes.
   - **TreeSet:** Stores elements in sorted, ascending order, making it useful when you need sorted elements.
2. **Performance:**
   - **HashSet:** Offers constant-time (O(1)) average-case complexity for basic operations like `add`, `remove`, and `contains`.
   - **TreeSet:** Provides guaranteed logarithmic-time (O(log n)) complexity for basic operations because it uses a Red-Black tree.
3. **Null Values:**
   - **HashSet:** Allows one `null` value.
   - **TreeSet:** Does not permit `null` values.
4. **Iteration:**
   - **HashSet:** The order of elements during iteration is not guaranteed and may vary.
   - **TreeSet:** Iterates over elements in sorted order.

Q.7 What is the difference between an Iterator and a ListIterator?

`Iterator` and `ListIterator` are interfaces in Java used to traverse collections, but they differ in several ways:

1. **Collections:**
   - **Iterator:** Can be used with any collection class that implements the `Iterable` interface.
   - **ListIterator:** Primarily designed for lists and can be used with classes that implement the `List` interface, like `ArrayList` and `LinkedList`.

2. **Movement:**
    - **Iterator:** Supports forward-only movement using methods like `next()` to move to the next element.
    - **ListIterator:** Allows bidirectional movement, meaning you can traverse both forward and backward through the list using methods like `next()` and `previous()`.
3. **Modification:**
    - **Iterator:** Supports removal of elements from a collection using the `remove()` method.
    - **ListIterator:** Offers more extensive modification capabilities, including adding and setting elements, in addition to removal.
4. **Index-Based Access:**
    - **Iterator:** Does not provide direct access to the index of the current element.
    - **ListIterator:** Provides methods like `nextIndex()` and `previousIndex()` to retrieve the index of the current element.
5. **Compatibility:**
    - **Iterator:** More commonly used due to its compatibility with a wider range of collections.
    - **ListIterator:** Used when you specifically need to work with lists and require bidirectional traversal and advanced modification options.

Q.8 What is the purpose of the Comparable interface?

The Comparable interface is used to provide a natural ordering for a class. It contains a single method compareTo() that compares the current object with another object of the same class and returns a negative integer, zero, or a positive integer depending on whether the current object is less than, equal to, or greater than the other object, respectively.

Q.9 What is the purpose of the java.util.concurrent package?

The java.util.concurrent package provides classes for concurrent programming, including thread pools, locks, atomic variables, and concurrent collections. It is designed to improve performance and scalability in multithreaded applications.

4. Exception Handling:

Q.1 What is an exception?

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Q.2 How does an exception propagate throughout the Java code?

When an exception occurs, it tries to locate the matching catch block. If the matching catch block is located, then that block is executed. Otherwise the exception propagates through the method call stack and goes into the caller method where the process of matching the catch block is performed. This happens until the matching catch block is found. In case the match is not found, the program gets terminated in the main method.

Q.3 What is the difference between checked and unchecked exceptions?

Checked exceptions are exceptions in Java that the compiler requires you to handle or declare in the method signature using the **`throws`** keyword. They are typically used for expected and recoverable errors.

Unchecked exceptions, often subclasses of **`RuntimeException`**, do not require explicit handling or declaration. They usually represent unexpected and unrecoverable errors, such as programming mistakes or system failures.

Q.4 What is the use of try-catch block in Java?

The try-catch block in Java is used to handle exceptions. It allows you to enclose code that might throw an exception within a "try" block and specify how to handle the exception in a "catch" block. This helps prevent program crashes and allows for graceful error handling, improving the robustness of Java applications.

Q.5 What is the difference between throw and throws?

In Java, "throw" is used to explicitly raise an exception within a method or code block, while "throws" is used in method declarations to indicate that the method may throw a specific type of exception, and the responsibility for handling that exception is deferred to the caller or higher-level code.

Q.6 What is the use of the finally block?

The "finally" block in Java is used to define a set of statements that will be executed regardless of whether an exception is thrown or not in a "try" block. It is commonly used to perform cleanup actions, such as closing files or releasing resources, ensuring that critical code is executed even in the presence of exceptions.

Q.7 What's the base class of all exception classes?

In Java, Java.lang.Throwable is the super class of all exception classes and all exception classes are derived from this base class.

Q.8 What is Java Enterprise Edition (Java EE)?

Java Enterprise Edition (Java EE) is a set of specifications and APIs for developing enterprise applications in Java. It includes a range of technologies, such as Servlets, JSPs, EJBs, JPA, JMS, and JNDI.

Q.9 What is the difference between a Servlet and a JSP?

A Servlet and a JSP (JavaServer Pages) are both Java-based technologies used for web development, but they serve different purposes:

1. **Servlet**:
    - A Servlet is a Java class that handles requests and generates responses on the server-side.
    - It is mainly used for controlling the business logic of a web application.
    - Servlets typically deal with low-level tasks like handling HTTP requests and responses directly.
2. **JSP (JavaServer Pages)**:
    - JSP is a technology that allows embedding Java code within HTML pages.
    - It is primarily used for creating dynamic web pages where Java code can be mixed with HTML markup.
    - JSP simplifies the process of building web interfaces by enabling the use of familiar HTML along with Java code snippets.

Q.10 What is the purpose of the Java Persistence API (JPA)?

The Java Persistence API (JPA) is a specification for object-relational mapping (ORM) in Java. It provides a set of interfaces and annotations for mapping Java objects to relational database tables and vice versa.

Q.11 What is the difference between stateful and stateless session beans?

Stateful session beans maintain a conversational state with the client, while stateless session beans do not. Stateful session beans are used for long-running conversations with a client, while stateless session beans are used for short-lived tasks.

5. Multithreading:

Q.1 What is a thread and what are the different stages in its lifecycle?

A thread is a lightweight process that can run concurrently with other threads in a program. Java thread life cycle has 5 stages: New, Runnable, Running, Non-Runnable(Blocked/ Waiting), Terminated.

Q.2 What is the difference between process and thread?

Process:

- A process is an independent and self-contained program that runs in its own memory space.
- Each process has its resources, including memory, file handles, and system data structures.
- Processes are heavyweight, consuming more system resources and taking more time to create and terminate.
- Processes are isolated from each other, and communication between processes typically involves inter-process communication (IPC) mechanisms like pipes or sockets.
- If one process crashes, it usually doesn't affect other processes.

Thread:

- A thread is a lightweight unit of a process that shares the same memory space as other threads in the same process.
- Threads within a process can communicate and share data more easily as they have shared memory.
- Threads are lightweight, consume fewer system resources, and are faster to create and terminate compared to processes.
- Threads within the same process can directly access each other's data and variables, making coordination easier.
- If one thread crashes, it can potentially affect other threads within the same process.

Q.3 What are the different types of thread priorities available in Java?

There are a total of 3 different types of priority available in Java. MIN_PRIORITY:Integer value 1. MAX_PRIORITY: Integer value 10.NORM_PRIORITY: Integer value 5.

Q.4 What is context switching in Java?

Context switching in Java is the process of switching from one thread to another by the operating system's scheduler. During context switching, the current thread's context, including its register values and program counter, are saved, and the next thread's context is restored.

Q.5 What is the difference between user threads and Daemon threads?

In Java, user threads have a specific life cycle and its life is independent of any other thread and are used for critical tasks. Daemon threads are basically referred to as a service provider that provides services and support to user threads. JVM does not wait for daemon threads to finish their tasks before termination., but waits for user threads.

Q.6 What is synchronization?

Synchronization in Java is a mechanism used to control access to shared resources (like variables or objects) in a multithreaded environment. It ensures that only one thread can access a synchronized block or method at a time, preventing data corruption or inconsistency caused by concurrent access.

Key points about synchronization:

- It helps in maintaining data integrity when multiple threads are involved.
- Java provides synchronized blocks and methods to define critical sections where only one thread can execute at a time.
- Synchronization can be applied to instance methods, static methods, or blocks of code.
- The `synchronized` keyword can be used with objects or classes to lock access.

- While synchronization prevents data races, it can also lead to performance bottlenecks if not used judiciously.

Q.7 What is a deadlock?

A deadlock is a situation where two or more threads are blocked waiting for each other to release the resources they need to proceed.

Q.8 What is the use of the wait() and notify() methods?

The `wait()` and `notify()` methods in Java are used for inter-thread communication and synchronization. Here's a brief summary of their purposes:

1. **wait():** This method is called on an object and instructs the current thread to temporarily release the lock it holds on that object. It puts the thread into a "waiting" state until another thread calls the `notify()` or `notifyAll()` method on the same object to wake it up. It's often used for synchronization and coordination between threads.
2. **notify():** The `notify()` method is used to wake up one of the threads that is waiting on the same object using the `wait()` method. If multiple threads are waiting, only one of them is chosen to be awakened. The choice is non-deterministic and depends on the JVM's implementation. To wake up all waiting threads, you can use the `notifyAll()` method.

Q.9 What is the difference between synchronized and volatile in Java?

Synchronized is used to provide exclusive access to a shared resource by allowing only one thread to access it at a time, while volatile is used to ensure visibility of changes made to a shared variable by guaranteeing that all threads see the same value.

Q.10 What is the purpose of the sleep() method in Java?

The sleep() method is used to pause the execution of a thread for a specified amount of time, allowing other threads to execute in the meantime.

Q.11 What is the difference between wait() and sleep() in Java?

In Java, both `wait()` and `sleep()` are methods used in multi-threading, but they serve different purposes:

1. **wait():** The `wait()` method is used for thread synchronization and inter-thread communication. It's called on an object, and it makes the current thread wait until another thread explicitly notifies it using the `notify()` or `notifyAll()` methods on the same object. It releases the lock on the object while waiting, allowing other threads to execute.
2. **sleep():** The `sleep()` method is used to introduce a pause or delay in the execution of a thread. It's not related to synchronization or inter-thread communication. When a thread calls `sleep()`, it goes into a "timed waiting" state for the specified duration, after which it automatically resumes execution. During this time, the thread retains the locks it holds.

Q.13 What is the difference between notify() and notifyAll() in Java?

Notify() is used to wake up a single thread that is waiting on an object, while notifyAll() is used to wake up all threads that are waiting on an object.

https://www.instagram.com/_sarthak_.2/