TOP 20 INTERVIEW QUESTIONS ON JAVA MULTITHREADING

1. What is Multithreading in Java?

Answer: Multithreading is a process of executing multiple threads simultaneously. It allows a program to perform multiple operations at the same time, improving performance and responsiveness.

---

2. How do you create a thread in Java?

Answer: A thread can be created by extending the `Thread` class and overriding the `run()` method or by implementing the `Runnable` interface and passing it to a `Thread` object.

---

3. What is the difference between `Thread` and `Runnable`?

Answer: `Thread` is a class, while `Runnable` is an interface. When you extend `Thread`, you cannot extend any other class, but by implementing `Runnable`, you can still extend other classes.

---

4. What are the different states of a thread in Java?

Answer: The states of a thread are:

- **New**: When a thread is created but not yet started.
- **Runnable**: When a thread is ready to run and is waiting for CPU time.
- **Blocked**: When a thread is blocked waiting for a resource.
- **Waiting**: When a thread is waiting indefinitely for another thread to perform a specific action.
- **Timed Waiting**: When a thread is waiting for a specified amount of time.
- **Terminated**: When a thread has finished execution.

---

5. What is the difference between `sleep()` and `wait()`?

Answer: `sleep()` pauses the current thread for a specified period of time without releasing any locks. `wait()` causes the current thread to wait until another thread calls `notify()` or `notifyAll()` on the same object, and it releases the lock.

---

6. How do you ensure thread safety in Java?

Answer: Thread safety can be ensured using synchronization, locks (like `ReentrantLock`), atomic classes from `java.util.concurrent.atomic`, and thread-safe collections from `java.util.concurrent`.

---

7. What is a synchronized block in Java?

Answer: A synchronized block restricts the access of a particular block of code to only one thread at a time, ensuring that shared resources are accessed in a thread-safe manner.

---

8. What is the difference between `synchronized` method and `synchronized` block?

Answer: A `synchronized` method locks the entire method for the current thread, whereas a `synchronized` block locks only a specific portion of the code within the method, allowing more granular control over synchronization.

---

9. What is a deadlock in Java?

Answer: A deadlock occurs when two or more threads are blocked forever, waiting for each other to release a resource. It usually happens when two threads have a circular dependency on resources.

---

10. How can you avoid deadlock in Java?

Answer: Deadlock can be avoided by following practices like avoiding nested locks, acquiring locks in a consistent order, and using try-lock mechanisms like `ReentrantLock.tryLock()`.

---

11. What is `volatile` keyword in Java?

Answer: The `volatile` keyword ensures that a variable's value is always read from the main memory, rather than being cached in a thread's local memory. It guarantees visibility of changes to variables across threads.

---

12. What is the difference between `notify()` and `notifyAll()`?

Answer: `notify()` wakes up a single thread that is waiting on the object's monitor, while `notifyAll()` wakes up all the threads that are waiting on the object's monitor. The choice between them depends on whether you want to notify one or all waiting threads.

---

13. What are the differences between `Executor` and `ExecutorService`?

Answer: `Executor` is a simple interface for launching tasks, whereas `ExecutorService` extends `Executor` and provides additional methods to manage the lifecycle of tasks, including shutdown and the ability to track the progress of tasks using `Future`.

---

14. What is a `Callable` interface in Java?

Answer: `Callable` is similar to `Runnable`, but it can return a result and throw a checked exception. It is used with `ExecutorService` to execute tasks that return values.

---

15. What is a thread pool and why is it used?

Answer: A thread pool is a pool of worker threads that are reused to execute multiple tasks. It improves performance by reducing the overhead of creating and destroying threads.

---

16. What is the `Future` interface in Java?

Answer: `Future` represents the result of an asynchronous computation. It provides methods to check if the computation is complete, retrieve the result, and cancel the computation.

---

17. What is the difference between `submit()` and `execute()` in Java?

Answer: `execute()` is used to submit a task for execution without expecting any result, while `submit()` can be used to submit a task and obtain a `Future` object representing the result of the task.

---

18. What is `ForkJoinPool` in Java?

Answer: `ForkJoinPool` is a special implementation of the `ExecutorService` that uses a work-stealing algorithm to efficiently manage tasks. It is used for parallel processing tasks, particularly those that can be broken down into smaller subtasks.

---

19. What is the difference between concurrency and parallelism?

Answer: Concurrency is about managing multiple tasks at the same time, allowing them to make progress independently. Parallelism involves executing multiple tasks simultaneously, often on multiple processors or cores.

---

20. What is a race condition, and how do you prevent it?

Answer: A race condition occurs when the outcome of a program depends on the sequence or timing of uncontrollable events, like the interleaving of thread execution. It can be prevented using synchronization, locks, or atomic variables to ensure that shared resources are accessed safely.
