MultiThreading Assignment

1. **Read.**
- Done
2. **Write a thread-safe singleton class**
3. **How to create a new thread(Please also consider Thread Pool approach)?**
- We could extend the Thread class, or implements Runnable / Callable interface.
4. **Difference between Runnable and Callable?**
- Runnable → run(), 无返回值，不上传错误
- Callable → call(), 有返回值，且上传错误
5. **What is the difference between t.start() and t.run()?**
- t.start() will start a new thread, while t.run() is just a normal method, it runs at the current thread.
6. **Which way of creating threads is better: Thread class or Runnable interface?**
- Mostly Runnable interface is better, since the class implementing the Runnable interface can still extend other classes, also it can be run by different threads/executors.
7. **What are the thread statuses?**
- There are 6 states of a thread
- NEW: Thread object is created, but not started yet
- RUNNABLE: Thread is ready to run or currently running
- BLOCKED: Thread is waiting to acquire a monitor lock
- WAITING: Thread is waiting indefinitely for another thread to act
- TIMED_WAITING: Thread is waiting for a specified time
- TERMINATED: Thread has finished execution
8. **Demonstrate deadlock and how to resolve it in Java code.**
- Deadlock occurs when two threads are waiting for each other.
- To fix it, we should control locking in the same global order
9. **How do threads communicate each other?**
- In Java, threads communicate by sharing data and coordinating execution. The JVM does not provide message passing like some languages; instead, communication happens through shared memory + synchronization mechanisms.
10. **What's the difference between class lock and object lock?**
- Object Lock (Instance Lock):
    - The monitor of a specific object instance
    - Each object has its own lock
- Class Lock (Static Lock):
    - The Class object (Counter.class)
    - There is one class lock per class, regardless of instances
11. **What is join() method?**
- join() tells the current thread to wait until another thread finishes.
12. **what is yield() method**

- yield() is a hint to the thread scheduler: I'm willing to pause and let other runnable threads run.

13. What is ThreadPool? How many types of ThreadPool? What is the TaskQueue in ThreadPool?

- A ThreadPool is a collection of pre-created worker threads that reuse threads to execute multiple tasks instead of creating a new thread every time.
- A thread pool consists of:
    - Worker Threads – execute tasks
    - Task Queue – holds waiting tasks
    - Thread Pool Manager – assigns tasks to threads
- The TaskQueue is a blocking queue that stores submitted tasks waiting to be executed.
- types of ThreadPool:
    - Fixed Thread Pool
    - Cached Thread Pool
    - Single Thread Executor
    - Scheduled Thread Pool
    - Work-Stealing Pool (Java 8+)

14. Which Library is used to create ThreadPool? Which Interface provide main functions of thread-pool?

- Java thread pools are provided by the java.util.concurrent library
- There are three key interfaces, with one primary interface you must know.

15. How to submit a task to ThreadPool?

- Submit a Runnable (no return value) using execute()
- Submit a Runnable with a Future, using submit(Runnable)
- Submit a Callable (returns a result), using submit(Callable<T>)
- Submitting multiple tasks at once, using invokeAll() – wait for all tasks

16. What is the advantage of ThreadPool?

- A ThreadPool provides better performance, resource control, and safer concurrency compared to creating threads manually.

17. Difference between shutdown() and shutdownNow() methods of executor

- shutdown() — graceful shutdown:
    - Stops accepting new tasks
    - Allows already submitted tasks to finish
    - Does NOT interrupt running threads
- shutdownNow() — immediate / forceful shutdown:
    - Stops accepting new tasks
    - Attempts to interrupt running tasks
    - Returns tasks that were never started

18. What is Atomic classes? How many types of Atomic classes? Give me some code example of Atomic classes and its main methods. when to use it?

- Atomic classes are thread-safe utility classes in java.util.concurrent.atomic that support lock-free, thread-safe operations on single variables using CAS (Compare-And-Swap) at the CPU level.
- Types of Atomic classes
    - Atomic primitive types
        - Used for single-value counters or flags
        - get()
        - set(value)
        - incrementAndGet()
        - getAndIncrement()
        - decrementAndGet()
        - addAndGet(delta)
        - compareAndSet(expect, update)
    - Atomic reference types
        - Used when you want to update object references atomically.
    - Atomic array types
        - Provide atomic operations on array elements
    - High-throughput adders (Java 8+)
        - Designed for heavy contention (many threads updating frequently).

19. What is the concurrent collections? Can you list some concurrent data structure (Thread-safe)

- Concurrent collections are thread-safe data structures provided by Java that allow multiple threads to access and modify them concurrently without external synchronization.
- Concurrent Map - ConcurrentHashMap
- Concurrent Queue & Deque - ConcurrentLinkedQueue / ConcurrentLinkedDeque
- Blocking Collections - BlockingQueue
- Concurrent Set & Copy-On-Write Collections - CopyOnWriteArrayList / CopyOnWriteArraySet

20. What kind of locks do you know? What is the advantage of each lock?

- Intrinsic Lock (Monitor Lock / synchronized)
    - Built into every Java object
    - Acquired using synchronized
    - Simple and easy to use
    - Automatically released (even on exception)
    - Supports wait() / notify()
    - Guaranteed correctness
- Explicit Lock (ReentrantLock)
    - A lock from java.util.concurrent.locks
    - Can tryLock() (avoid deadlock)

    - Supports timeout
    - Optional fairness
    - Multiple Condition objects
    - More flexible than synchronized
- Read–Write Lock (ReentrantReadWriteLock)
    - Separates read and write access.
    - Multiple readers at the same time
    - Better performance for read-heavy workloads
    - Writers still get exclusive access
- Spin Lock
    - A thread spins (busy-waits) instead of blocking.
    - Very fast for short critical sections
    - No context switch overhead

21. What is future and completableFuture? List some main methods of ComplertableFuture.

- Future: represents the result of an asynchronous computation.
    - Returned when a task is submitted to a thread pool
    - Allows you to:
        - check if the task is done
        - wait for the result
        - cancel the task
- CompletableFuture: is an advanced, non-blocking, asynchronous API introduced in Java 8.
    - Implements Future
    - Supports functional-style chaining
    - Allows callbacks instead of blocking
    - Has powerful exception handling
    - Can combine multiple async tasks
- Main methods of CompletableFuture
    - Creation methods
        - runAsync(Runnable)
        - supplyAsync(Supplier<T>)
        - completedFuture(T value)
    - Transformation (map-like)
        - thenApply(Function<T, R>)
        - thenAccept(Consumer<T>)
        - thenRun(Runnable)
    - Chaining async tasks
        - thenCompose(Function<T, CompletableFuture<R>>)
    - Combine multiple futures

    - thenCombine(cf2, BiFunction)
    - allOf(cf1, cf2, cf3)
    - anyOf(cf1, cf2)
- Exception handling ⭐
    - exceptionally(Function<Throwable, T>)
    - handle(BiFunction<T, Throwable, R>)
    - whenComplete(BiConsumer<T, Throwable>)
- Completion & control
    - get()
    - join()
    - complete(value)
    - completeExceptionally(ex)
    - cancel()