Questions on CompletableFuture

1. Basic Asynchronous Task

Write a program that uses `CompletableFuture` to:

1. Perform a computation asynchronously to square a number.
2. Once the computation is done, print the result to the console using `thenAccept`.

---

2. Chaining Tasksn

Create a program where:

1. A `CompletableFuture` fetches a string asynchronously (e.g., a username).
2. Another task converts the string to uppercase.
3. Finally, print the result to the console.

---

3. Combining Two Futures

Write a program to:

1. Fetch two numbers asynchronously using two separate `CompletableFuture`s.
2. Combine the results of both numbers to compute their sum.
3. Print the sum to the console.

---

4. Handling Exceptions

Write a program that:

1. Asynchronously fetches a value (simulate a delay).
2. Simulates an exception in the task.
3. Use `exceptionally` or `handle` to gracefully handle the exception and provide a fallback value.

---

5. Parallel API Calls

Prepared by Dewansh Nigam

Suppose you are fetching data from two APIs:

1. Create two `CompletableFuture`s that simulate API calls by returning data after a delay.
2. Use `allOf` to wait for both tasks to complete.
3. Once both are completed, combine their results and print them.

---

6. Run Multiple Independent Tasks

Write a program where:

1. Five tasks are executed in parallel using `CompletableFuture.runAsync`.
2. Each task prints its completion message.
3. Wait for all tasks to finish using `allOf`.

---

7. Pipeline with Dependent Tasks

Simulate the following:

1. Fetch a user ID asynchronously.
2. Use the ID to fetch user details asynchronously.
3. Once the user details are fetched, print them to the console.

---

8. Timeout Handling

Write a program that:

1. Executes a task asynchronously using `supplyAsync`.
2. If the task does not complete within 2 seconds, use `completeOnTimeout` to provide a default value.

---

9. Fetch the First Result

You are querying multiple servers for the same data. Write a program that:

1. Simulates querying three servers using three `CompletableFuture`s.
2. Uses `anyOf` to return the result of the first server that responds.

Prepared by Dewansh Nigam

3. Print the result to the console.

---

10. Dependent Futures with Exception Handling

Create a program where:

1. A task fetches a number asynchronously.
2. Another task divides a fixed number by the fetched number.
3. Handle the scenario where the fetched number is 0 using `exceptionally` or `handle`.

---

11. Progress Monitoring

Write a program where:

1. A long-running computation (e.g., factorial of a large number) is executed asynchronously.
2. Periodically print progress updates using another thread or scheduled tasks.

---

12. Recursive Async Tasks

Simulate a scenario where:

1. You calculate the nth Fibonacci number using recursive `CompletableFuture`s.
2. Use asynchronous tasks to compute each Fibonacci number in parallel.

---

13. Conditional Execution

Write a program that:

1. Fetches a number asynchronously.
2. If the number is greater than 10, computes its square.
3. If the number is less than or equal to 10, computes its cube.
4. Print the result.

---

14. Parallel Reduction

Suppose you have a list of integers:

1. Use `CompletableFuture` to compute the square of each integer in parallel.
2. Reduce the results to their sum and print the result.

---

15. Async File Reading

Write a program that:

1. Reads the contents of two files asynchronously.
2. Combines their contents into a single string.
3. Writes the combined string to a new file asynchronously.

---

16. Delayed Execution

Simulate a task scheduler:

1. Use `CompletableFuture.delayedExecutor` to execute a task after a 3-second delay.
2. Print a message indicating the task was executed.

---

17. Real-World Simulation: Order Processing

Simulate an e-commerce order processing system:

1. Fetch user details asynchronously.
2. Fetch order details asynchronously based on the user.
3. Combine user and order details to print the complete order information.

---

18. Progress Bar Simulation

Create a program that:

1. Simulates a task running for 10 seconds asynchronously.
2. Updates a progress bar in the console every second using another `CompletableFuture`.

---

19. API Aggregation

Simulate a scenario where:

1. You call three APIs (or simulate with `CompletableFuture`).
2. Combine the results of the three APIs into a single JSON-like object.
3. Print the aggregated result.

---

20. Dynamic Number of Tasks

Write a program where:

1. You are given a list of URLs.
2. Use `CompletableFuture` to fetch the content of each URL in parallel (simulate with delays).
3. Wait for all results to complete and then print the combined results.

Bonus Problem: Asynchronous Retry Mechanism

Write a program where:

1. A task tries to fetch data from a simulated API (can fail randomly).
2. If the task fails, retry it up to 3 times with a delay between retries.
3. If all retries fail, print an error message.

Questions on Multithreading

1. Print Numbers Alternately

Write a program with two threads:

1. One thread prints odd numbers from 1 to 10.
2. The other thread prints even numbers from 1 to 10.
3. Ensure the numbers are printed in sequence (1, 2, 3, 4...).

---

2. Producer-Consumer Problem

Implement the classic producer-consumer problem:

1. Use a shared buffer with a fixed size.
2. A producer thread generates random numbers and adds them to the buffer.
3. A consumer thread consumes numbers from the buffer and prints them.
4. Use proper synchronization to avoid race conditions.

---

3. Dining Philosophers

Simulate the Dining Philosophers problem:

1. Five philosophers alternate between eating and thinking.
2. They share five forks, and each philosopher needs two forks to eat.
3. Avoid deadlock and starvation using proper thread synchronization.

---

4. Deadlock Simulation

Write a program to simulate a deadlock:

1. Create two threads where each thread tries to lock two shared resources in a different order.
2. Demonstrate how deadlock occurs and explain how to resolve it.

---

5. Print Fibonacci Numbers Using Threads

1. Create three threads to generate Fibonacci numbers.
2. Each thread computes a part of the sequence, and the final sequence is constructed by combining their results.

---

6. Countdown Timer

Write a program to implement a countdown timer:

1. Create a thread that counts down from 10 to 0.
2. Print the countdown value at 1-second intervals.
3. Stop the timer prematurely using another thread.

---

7. Parallel Sum of Array

1. Divide an array of integers into chunks.
2. Use multiple threads to calculate the sum of each chunk in parallel.
3. Combine the results to calculate the total sum.

---

8. Thread-safe Singleton

1. Implement a thread-safe Singleton class using `synchronized`.
2. Demonstrate that only one instance is created even when accessed by multiple threads simultaneously.

---

9. Thread-safe Counter

1. Create a class `Counter` with methods `increment` and `getValue`.
2. Use multiple threads to increment the counter 1000 times each.
3. Ensure the counter value is correct by making it thread-safe.

---

10. Merge Sorted Arrays in Parallel

1. Given two sorted arrays, merge them into a single sorted array using multiple threads.
2. Split the work across threads to handle different sections of the arrays.

---

11. Thread-safe Data Structure

1. Implement a thread-safe version of a queue.
2. Multiple producer and consumer threads should be able to safely enqueue and dequeue elements.

---

12. Parallel Matrix Multiplication

1. Write a program to multiply two matrices in parallel.
2. Use one thread for each row of the resulting matrix.

---

13. File Reading and Processing

1. Divide a large text file into chunks.
2. Use multiple threads to read and process each chunk.
3. Combine the processed results at the end.

---

14. Print "Ping-Pong"

1. Create two threads: one prints "Ping" and the other prints "Pong".
2. Ensure they alternate correctly to print "Ping Pong Ping Pong...".

---

15. Thread Interruption

1. Write a program where a thread performs a long-running task.
2. Interrupt the thread from another thread and handle the interruption gracefully.

---

16. Thread Pool Implementation

1. Implement a simple thread pool.
2. Create a fixed number of threads that pick tasks from a shared queue and execute them.

---

17. Traffic Light Simulation

1. Simulate a traffic light system using threads.
2. Each thread controls one traffic light and changes its state (Red, Green, Yellow) at regular intervals.

---

18. Parallel Word Count

1. Count the occurrences of each word in a large text file.
2. Use multiple threads to process different parts of the file.

---

19. Semaphore Usage

1. Simulate a parking lot with a limited number of parking spaces.

2. Use a `Semaphore` to manage access to the parking spaces.

---

20. Multithreaded Web Crawler

1. Write a web crawler that downloads and processes web pages.
2. Use multiple threads to fetch pages in parallel.

---

21. Custom Thread Scheduling

1. Create a custom thread scheduler.
2. Assign priority to threads and ensure they are executed based on priority.

---

22. Barrier Synchronization

1. Simulate a race with multiple runners (threads).
2. All threads should start running at the same time after a signal (use `CyclicBarrier`).

---

23. Bank Account Transfer

1. Implement a `BankAccount` class with methods for deposit and withdrawal.
2. Use multiple threads to simulate concurrent transfers between accounts.
3. Ensure thread safety to avoid inconsistent balances.

---

24. Read-Write Lock

1. Implement a thread-safe shared data structure with read-write lock.
2. Multiple reader threads should read data concurrently.
3. Only one writer thread should be allowed to write at a time.

---

25. Concurrent Sorting

1. Divide an array into sections.
2. Use multiple threads to sort each section in parallel.

3. Merge the sections to produce the final sorted array.

Bonus Problem: Asynchronous Pipeline

1. Simulate a production pipeline with three stages:
    - Stage 1: Generate raw materials.
    - Stage 2: Process the materials.
    - Stage 3: Package the final product.
2. Use three threads where each thread represents a stage. Pass data between stages using a thread-safe queue.