**Debugging and Unit Testing**

**1. Debugging a Multi-Threaded Race Condition (Breakpoints, Step Execution)**

**Problem Statement:**

* You are working on a **multi-threaded application** where multiple threads update a shared counter.
* The counter sometimes produces **incorrect results** due to a **race condition**.
* Use **breakpoints** in your IDE to:
  + Pause execution when multiple threads access the counter.
  + Step through execution to identify incorrect updates.
  + Use **watchpoints** to track changes to the counter variable.
* Fix the issue using **synchronization** or AtomicInteger.

**2. Testing a REST API Service (JUnit, Assertions, Mocking)**

**Problem Statement:**

* Develop and test a **REST API service** for a **User Management System** with endpoints like:
  + POST /users (create user)
  + GET /users/{id} (fetch user)
  + PUT /users/{id} (update user)
* Use **JUnit and assertions** to:
  + Write unit tests for the service methods.
  + Mock dependencies like **database access** using **Mockito**.
  + Validate responses using assertions (assertEquals, assertNotNull).
* Implement **test coverage** analysis to ensure all cases are tested.

**3. Debugging Memory Leaks in a Java Application (Heap Analysis, Profiling)**

**Problem Statement:**

* Your application **crashes due to OutOfMemoryError** after running for a long time.
* Use **heap dump analysis and profiling tools** (VisualVM, Eclipse MAT) to:
  + Identify objects that are **not garbage collected**.
  + Track **unreleased references** causing memory leaks.
  + Use breakpoints in constructors/destructors to verify object lifecycles.
* Fix the issue by ensuring proper **resource cleanup** and using **weak references** where needed.

**4. Unit Testing a Banking System (JUnit, Parameterized Tests, Exception Handling)**

**Problem Statement:**

* Implement and test a **Banking System** with functionalities:
  + deposit(double amount)
  + withdraw(double amount)
  + transferFunds(Account to, double amount)
* Use **JUnit parameterized tests** to test multiple valid/invalid inputs.
* Write unit tests for **exception handling** (e.g., withdrawing more than balance).
* Use **Mockito** to mock external dependencies (e.g., database transactions).

**5. Debugging a Recursive Algorithm (Step Execution, Logging, Breakpoints)**

**Problem Statement:**

* A **recursive algorithm for calculating Fibonacci numbers** sometimes produces incorrect results.
* Debug the recursion using:
  + **Step execution** to track how the recursion tree expands and backtracks.
  + **Breakpoints at the base case** to check if it terminates correctly.
  + **Logging** to print intermediate values.
* Optimize the function using **memoization** and **iterative approaches**.