Core Java-Assesment8

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

class Counter {

private int count = 0;

// Method to increment the counter

public void increment() {

count++;

}

// Get current count

public int getCount() {

return count;

}

}

public class RaceConditionExample {

public static void main(String[] args) {

Counter counter = new Counter();

// Thread 1

Thread t1 = new Thread(() -> {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

});

// Thread 2

Thread t2 = new Thread(() -> {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

});

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

// The result is incorrect due to race condition

System.out.println("Final count: " + counter.getCount());

}

}

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

import org.junit.jupiter.api.Test;import org.mockito.Mockito;import static org.mockito.Mockito.\*;import static org.junit.jupiter.api.Assertions.\*;

class ApiService {

public String fetchDataFromApi(String url) {

// Simulate an API call

return "Response from " + url;

}

}

public class ApiServiceTest {

@Test

public void testFetchDataFromApi() {

// Mocking the ApiService class

ApiService mockApiService = Mockito.mock(ApiService.class);

// Define behavior of the mocked method

when(mockApiService.fetchDataFromApi("http://example.com")).thenReturn("Mocked response");

// Test the mocked method

String response = mockApiService.fetchDataFromApi("http://example.com");

// Assertions

assertEquals("Mocked response", response);

}

}

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

import java.util.ArrayList;import java.util.List;

public class MemoryLeakExample {

public static void main(String[] args) {

List<byte[]> memoryLeakList = new ArrayList<>();

// Simulate memory leak by continuously adding large objects to the list

for (int i = 0; i < 1000; i++) {

memoryLeakList.add(new byte[1024 \* 1024]); // 1MB object

}

// Ideally, you'd use a profiling tool to analyze memory usage

System.out.println("Simulated memory leak. Use a profiler to track memory usage.");

}

}

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

import org.junit.jupiter.api.Test;import org.junit.jupiter.api.Assertions;

class BankAccount {

private int balance = 1000;

public void withdraw(int amount) throws IllegalArgumentException {

if (amount > balance) {

throw new IllegalArgumentException("Insufficient balance");

}

balance -= amount;

}

public int getBalance() {

return balance;

}

}

public class BankAccountTest {

@Test

public void testWithdrawSuccess() {

BankAccount account = new BankAccount();

account.withdraw(500);

Assertions.assertEquals(500, account.getBalance());

}

@Test

public void testWithdrawFailure() {

BankAccount account = new BankAccount();

// Testing exception

Assertions.assertThrows(IllegalArgumentException.class, () -> account.withdraw(1500));

}

}

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

public class Factorial {

public static int factorial(int n) {

if (n == 0) {

return 1;

}

System.out.println("Factorial of " + n + ": " + n \* factorial(n - 1)); // Logging

return n \* factorial(n - 1);

}

public static void main(String[] args) {

int result = factorial(5);

System.out.println("Factorial of 5: " + result);

}

}