Core Java-Assessment7

### ****1. Memory Leak Detector and Fixer (Garbage Collection, JVM Memory Model)****

import java.util.\*;

public class MemoryLeakDetector {

public static void main(String[] args) {

List<Object> list = new ArrayList<>();

// Simulating a memory leak by continuously adding new objects to the list

for (int i = 0; i < 100000; i++) {

list.add(new Object());

}

// Suggestion: The above code simulates a memory leak (continuous object creation without removal)

// JVM's garbage collector will eventually clean up unused objects (not used in list anymore).

// To fix this, you would avoid unnecessary object retention like above.

// Force garbage collection (for demonstration)

System.gc();

System.out.println("Garbage collection completed. Potential memory leak fixed.");

}

}

### ****2. Multi-threaded Order Processing System (Threads, Synchronization)****

class OrderProcessor extends Thread {

private String orderId;

public OrderProcessor(String orderId) {

this.orderId = orderId;

}

@Override

public void run() {

synchronized (this) {

// Simulate processing order

System.out.println("Processing order: " + orderId);

try {

Thread.sleep(1000); // Simulate processing time

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Order " + orderId + " processed");

}

}

}

public class MultiThreadedOrderProcessing {

public static void main(String[] args) {

OrderProcessor order1 = new OrderProcessor("A101");

OrderProcessor order2 = new OrderProcessor("A102");

order1.start();

order2.start();

}

}

### ****3. Bank Transaction System (Threads, Synchronization, Deadlock Prevention)****

class Account {

private int balance = 1000;

public synchronized void withdraw(int amount) {

if (balance >= amount) {

balance -= amount;

System.out.println("Withdrawal successful. New balance: " + balance);

} else {

System.out.println("Insufficient balance.");

}

}

public synchronized void deposit(int amount) {

balance += amount;

System.out.println("Deposit successful. New balance: " + balance);

}

}

public class BankTransactionSystem {

public static void main(String[] args) {

Account account1 = new Account();

Account account2 = new Account();

// Thread 1: Transfer money from account1 to account2

Thread t1 = new Thread(() -> {

synchronized (account1) {

account1.withdraw(100);

synchronized (account2) {

account2.deposit(100);

}

}

});

// Thread 2: Transfer money from account2 to account1

Thread t2 = new Thread(() -> {

synchronized (account2) {

account2.withdraw(50);

synchronized (account1) {

account1.deposit(50);

}

}

});

t1.start();

t2.start();

}

}