Task 4: Synchronized Blocks and Methods

Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.

```
package com.wipro.model;

class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public synchronized void deposit(double amount) {

System.out.println(Thread.currentThread().getName() + " is depositing " + amount);

balance += amount;

System.out.println("New balance after deposit by " + Thread.currentThread().getName() + ": " +
balance);

}//deposit

public synchronized void withdraw(double amount) {

if (balance >= amount) {

System.out.println(Thread.currentThread().getName() + " is withdrawing " + amount);

balance -= amount;

System.out.println("New balance after withdrawal by " + Thread.currentThread().getName() + ": " +
balance);

} else {

System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + " but
balance is insufficient.");

}

}

}//withdraw

public class BankAccountDemo {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000);

Thread thread1 = new Thread(() -> {

account.deposit(500);

}, "Thread-1");
Thread thread2 = new Thread(() -> {
account.withdraw(200);
}, "Thread-2");
Thread thread3 = new Thread(() -> {
account.deposit(100);
}, "Thread-3");
Thread thread4 = new Thread(() -> {
account.withdraw(700);
}, "Thread-4");
thread1.start();
thread2.start();
thread3.start();
thread4.start();
}
}
```