DAY 23:

ASSIGNMENTS 4:

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.

ANSWER:

BankAccount Class

This class represents a bank account with methods for depositing and withdrawing money. The methods are synchronized to ensure thread safety.

java

```
class BankAccount {
    private int balance;

    public BankAccount(int initialBalance) {
        this.balance = initialBalance;
    }

    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited " + amount + ". New balance: " + balance);
    }

    public synchronized void withdraw(int amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ". New balance: " + balance);
        } else {
            System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + " but
only " + balance + " available.");
        }
    }

    public synchronized int getBalance() {
        return balance;
    }
}

class DepositRunnable implements Runnable {
    private final BankAccount account;
    private final int amount;

    public DepositRunnable(BankAccount account, int amount) {
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            account.deposit(amount);
            try {
                Thread.sleep(100); // Simulate time taken for the deposit operation
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
class WithdrawRunnable implements Runnable {
    private final BankAccount account;
    private final int amount;

    public WithdrawRunnable(BankAccount account, int amount) {
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            account.withdraw(amount);
            try {
                Thread.sleep(150); // Simulate time taken for the withdrawal operation
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class BankAccountDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);

        Thread depositThread1 = new Thread(new DepositRunnable(account, 200), "DepositThread1");
        Thread depositThread2 = new Thread(new DepositRunnable(account, 300), "DepositThread2");
        Thread withdrawThread1 = new Thread(new WithdrawRunnable(account, 150),
"WithdrawThread1");
```

```
        Thread withdrawThread2 = new Thread(new WithdrawRunnable(account, 400),
"WithdrawThread2");

        depositThread1.start();
        depositThread2.start();
        withdrawThread1.start();
        withdrawThread2.start();

        try {
            depositThread1.join();
            depositThread2.join();
            withdrawThread1.join();
            withdrawThread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Final balance: " + account.getBalance());
    }
}
```

Explanation:

1. BankAccount Class:

- This class has a private balance field representing the account balance.
- The deposit method increases the balance. It is synchronized to ensure that only one thread can execute it at a time.
- The withdraw method decreases the balance if sufficient funds are available. It is also synchronized to prevent race conditions.
- The getBalance method returns the current balance and is synchronized for consistency.

2. DepositRunnable Class:

- Implements the Runnable interface.

- The run method calls the deposit method of the BankAccount class multiple times, simulating multiple deposit operations.

3. WithdrawRunnable Class:

- Implements the Runnable interface.
- The run method calls the withdraw method of the BankAccount class multiple times, simulating multiple withdrawal operations.

4. BankAccountDemo Class:

- Creates an instance of BankAccount with an initial balance.
- Creates multiple threads to perform deposits and withdrawals concurrently.
- Starts the threads and waits for them to finish using the join method.
- Prints the final balance of the account.

Running the Program:

When you run this program, you will see the threads performing deposit and withdrawal operations on the shared BankAccount object. The synchronized methods ensure that the account balance is updated correctly without race conditions, and the final balance is printed at the end.