НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №3

із дисципліни «Технології паралельних та розподілених обчислень»

на тему «Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи»

Підготував:

Студент ФІОТ, гр. ІТ-83,

Троян О.С.

Перевірив:

пос. Дифучина О. Ю.

Київ 2020

Мета роботи: Розробка паралельних програм з використанням механізмів синхронізації

Завдання:

1. Реалізуйте програмний код, даний у лістингу, та протестуйте його при різних значеннях параметрів. Модифікуйте програму, використовуючи методи управління потоками, так, щоб її робота була завжди коректною. Запропонуйте три різних варіанти управління.

   30 балів

2. Реалізуйте приклад Producer-Consumer application (див. https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html). Модифікуйте масив даних цієї програми, які читаються, у масив чисел заданого розміру (100, 1000 або 5000) та протестуйте программу. Зробіть висновок про правильність роботи програми.

   20 балів

3. Реалізуйте роботу електронного журналу групи, в якому зберігаються оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і його 3 асистенти виставляють оцінки з дисципліни за 100-бальною шкалою.

   40 балів

4. Зробіть висновки про використання методів управління потоками в java.

   10 балів

Хід роботи

1. Я реалізував програмний код даний у лістингу та запропонував 3 способи синхронізувати транзакції так, щоб сума рахунку усіх користувачів завжди була однакова.

```
public interface IBank {
    void transfer(int fromAccount, int toAccount, int amount) throws InterruptedException;
    int size();
}
```

```
public class Bank implements IBank{
    public static final int NTEST = 10000;
    private final int[] accounts;
    private long ntransacts = 0;

    public Bank(int n, int initialBalance){
        accounts = new int[n];
        int i;
        for (i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
        ntransacts = 0;
    }

    public void transfer(int from, int to, int amount)
            throws InterruptedException{
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0)
            test();
    }
    public void test(){
        int sum = 0;
        for (int i = 0; i < accounts.length; i++)
            sum += accounts[i] ;
        System.out.println("Transactions:" + ntransacts
                + " Sum: " + sum);
    }

    public int size(){
        return accounts.length;
    }
}
```

```
public static void startTask(int naCCounts, int initial_balance){
    //IBank b = new SynchronizedBank(naCCounts, initial_balance); //
    var b = new Bank(naCCounts, initial_balance);
    //var b = new ReentrantLockBank(naCCounts, initial_balance);
    int i;
    for (i = 0; i < naCCounts; i++){
        var t = new TransferThread(b, i,
                initial_balance);
        t.setPriority(Thread.NORM_PRIORITY + i % 2);
        t.start ();
    }
}
```

При звичайній роботі програми кожні десять тисяч транзакцій частина з них не коректно працює і ми отримаємо рахунок менший с кожною трансакцією хоча сума не повинна змінюватись, так як ми передаємо другому користувачу стільки ж, скільки віднімаємо у першого.

```
Transactions:10002 Sum: 99916
Transactions:20007 Sum: 99860
Transactions:30001 Sum: 99629
Transactions:40004 Sum: 99606
```

1.1. Синхронізація методів

```java
public class SynchronizedBank implements IBank{
    public static final int NTEST = 10000;
    private final int[] accounts;
    private long ntransacts = 0;

    public SynchronizedBank(int n, int initialBalance){
        accounts = new int[n];
        int i;
        for (i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
        ntransacts = 0;
    }

    public synchronized void transfer(int from, int to, int amount)
            throws InterruptedException{
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0)
            test();
    }
    public void test(){
        int sum = 0;
        for (int i = 0; i < accounts.length; i++)
            sum += accounts[i] ;
        System.out.println("Transactions:" + ntransacts
                + " Sum: " + sum);
    }

    public int size(){
        return accounts.length;
    }
}
```

Результат

```
Transactions:10000 Sum: 100000
Transactions:20000 Sum: 100000
Transactions:30000 Sum: 100000
Transactions:40000 Sum: 100000
```

1.2. Синхронізація через об'єкт

```java
public class SynchronizedObjectBank implements IBank{
    public static final int NTEST = 10000;
    private final int[] accounts;
    private long ntransacts = 0;
    private Object object = new Object();

    public SynchronizedObjectBank(int n, int initialBalance){
        accounts = new int[n];
        int i;
        for (i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
        ntransacts = 0;
    }

    @Override
    public void transfer(int from, int to, int amount)
            throws InterruptedException{
        synchronized (object){
            accounts[from] -= amount;
            accounts[to] += amount;
            ntransacts++;
            if (ntransacts % NTEST == 0)
                test();
        }
    }

    public void test(){
        int sum = 0;
        for (int i = 0; i < accounts.length; i++)
            sum += accounts[i] ;
        System.out.println("Transactions:" + ntransacts
                + " Sum: " + sum);
    }

    public int size(){
        return accounts.length;
    }
}
```

Результат

```
Transactions:10000 Sum: 100000
Transactions:20000 Sum: 100000
Transactions:30000 Sum: 100000
Transactions:40000 Sum: 100000
```

Обидва варіанти блокують монітор тим самим не дають можливість роботи з об'єктом до кінця операції.

1.3. Локер

```java
public class ReentrantLockBank implements IBank{
    public static final int NTEST = 10000;
    private final int[] accounts;
    private long ntransacts = 0;
    private final ReentrantLock _locker = new ReentrantLock();

    public ReentrantLockBank(int n, int initialBalance){
        accounts = new int[n];
        int i;
        for (i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
        ntransacts = 0;
    }

    public void transfer(int from, int to, int amount) throws InterruptedException{
        try {
            _locker.lock();

            accounts[from] -= amount;
            accounts[to] += amount;
            ntransacts++;
            if (ntransacts % NTEST == 0)
                test();
        }finally {
            _locker.unlock();
        }
    }
    public void test(){
        int sum = 0;
        for (int i = 0; i < accounts.length; i++)
            sum += accounts[i] ;
        System.out.println("Transactions:" + ntransacts
                + " Sum: " + sum);
    }

    public int size(){
        return accounts.length;
    }
}
```

Результат

```
Transactions:10000 Sum: 100000
Transactions:20000 Sum: 100000
Transactions:30000 Sum: 100000
```

Всі 3 варіанта забезпечують точну роботу з транзакціями.

2. Я реалізував приклад Producer-Consumer application та змінив тип даних що передаються на масив чисел.

```java
public Producer(Drop drop, int[] importantInfo) {
    this.drop = drop;
    this.importantInfo = importantInfo;
}

public void run() {
    Random random = new Random();

    for (int i = 0;
         i < importantInfo.length;
         i++) {
        drop.put(String.valueOf(importantInfo[i]));
        try {
            Thread.sleep(random.nextInt( bound: 5));
        } catch (InterruptedException e) {}
    }
    drop.put( message: "DONE");
}
```

3. Я створив програму що імітує журнал в який записують свої оцінки лектор та три асистенти з трьох різних потоків. Я взяв для експерименту 3 тижня, а отже і ітерацій та три групи по 15 студентів. Після синхронізації методу запис в журнал став синхронізований.

```java
public void addMark(String groupName, Integer studentName, String mark) {
    synchronized (this.groups.get(groupName).groupList.get(studentName)) {
        this.groups.get(groupName).groupList.get(studentName).add(mark);
    }
}
```

```
Group name:     ІП
Student    1    -           69: As2(w1)          90: As1(w1)          42: As1(w2)          99: As2(w2)          18: L1(w1)
20: As3(w1)          17: As1(w3)          89: As2(w3)          17: L1(w2)           5: As3(w2)           0: L1(w3)            60: As
3(w3)
Student    2    -           64: As2(w1)          79: As1(w1)          45: As1(w2)          65: As2(w2)          29: L1(w1)
85: As3(w1)           1: As1(w3)          88: As2(w3)          83: L1(w2)           15: As3(w2)          42: L1(w3)           82: As
3(w3)
```

Висновок: В java є велика кількість способів управлінням потоками для різних випадків.
При потребі в безпечних у потоках змінних без синхронізації ми можемо використовувати атомарні змінні та методи.
При потребі в доступі декількох потоків до одного об’єкту за синхронізації доступу до нього ми використовуємо блокований об’єкт.
Якщо нам треба послідовне виконання методів потоку то ми використовуємо синхронізовані методи.

Але ми повинні пам’ятати що синхронізація роботи потоків сповільнює виконання програми, тому ми повинні намагатися використовувати синхронізацію тільки для методів, які виконують зчитування спільних даних та їх оновлення. Такі методи повинні бути якомога коротшими.