

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З комп'ютерного практикуму № 3 з дисципліни
«Технології паралельних обчислень»

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

Виконав(ла)

ІП-15 Мешков Андрій

(шифр, прізвище, ім'я, по батькові)

Перевірів

Дифучина О. Ю.

(шифр, прізвище, ім'я, по батькові)

Київ 2024

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 програму. 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

Завдання 1

Завдання полягає в реалізації та тестуванні коду асинхронного банківського додатку, який використовує багатопотоковість для проведення трансакцій між рахунками. Також необхідно модифікувати програму, використовуючи методи управління потоками, щоб забезпечити коректну роботу програми, та запропонувати три різні варіанти управління потоками.

Три різні варіанти управління потоками.

Синхронізовані методи (synchronized methods) - У цьому варіанті використовується ключове слово `synchronized`, щоб забезпечити ексклюзивний доступ до методу `transfer`.

Блокування (ReentrantLock) - У цьому варіанті використовується `ReentrantLock`, щоб контролювати доступ до методу `transfer`.

Блокування на рівні об'єкта (synchronized block) - У цьому варіанті використовується блокування на рівні об'єкта, щоб синхронізувати доступ до методу `transfer`.

Лістинг коду:

TransferThread.java

```
package task1;

public class TransferThread extends Thread {

    private Bank bank;
    private int fromAccount;
    private int maxAmount;
    private static final int REPS = 1000;
    public TransferThread(Bank b, int from, int max){
        bank = b;
        fromAccount = from;
        maxAmount = max;
    }
    @Override
    public void run(){
        while (true) {
            for (int i = 0; i < REPS; i++) {
                int toAccount = (int) (bank.size() * Math.random());
                int amount = (int) (maxAmount * Math.random()/REPS);
```

```
        bank.transfer(fromAccount, toAccount, amount);  
        bank.transferSyncMethod(fromAccount, toAccount, amount):  
    synchronized (bank) {  
        bank.transfer(fromAccount, toAccount, amount);  
    }  
    bank.transferLock(fromAccount, toAccount, amount);  
}  
  
}  
  
}
```

Bank.java

```
package task1;

import java.util.concurrent.locks.ReentrantLock;

public class Bank {

    public static final int NTEST = 10000;
    private final int[] accounts;
    private long ntransacts = 0;

    private final ReentrantLock locker = new ReentrantLock();
    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) {
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0)
            test();
    }

    public synchronized void transferSyncMethod(int from, int to, int amount) {
        transfer(from, to, amount);
    }

    public void transferLock(int from, int to, int amount) {
        locker.lock();
        try {
            transfer(from, to, amount);
        }
        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;
}
}

```

AsyncBankTest.java

```

package task1;

public class AsyncBankTest {
    public static final int NACCOUNTS = 100;
    public static final int INITIAL_BALANCE = 10000;
    public static void main(String[] args) {
        Bank b = new Bank(NACCOUNTS, INITIAL_BALANCE);
        int i;
        for (i = 0; i < NACCOUNTS; i++){
            TransferThread t = new TransferThread(b, i,
                INITIAL_BALANCE);
            t.setPriority(Thread.NORM_PRIORITY + i % 2);
            t.start() ;
        }
    }
}

```

Результат:

```

Transactions:37770000 Sum: 1000000
Transactions:37780000 Sum: 1000000
Transactions:37790000 Sum: 1000000
Transactions:37800000 Sum: 1000000
Transactions:37810000 Sum: 1000000
Transactions:37820000 Sum: 1000000
Transactions:37830000 Sum: 1000000
Transactions:37840000 Sum: 1000000
Transactions:37850000 Sum: 1000000
Transactions:37860000 Sum: 1000000
Transactions:37870000 Sum: 1000000
Transactions:37880000 Sum: 1000000
Transactions:37890000 Sum: 1000000
Transactions:37900000 Sum: 1000000
Transactions:37910000 Sum: 1000000
Transactions:37920000 Sum: 1000000
Transactions:37930000 Sum: 1000000
Transactions:37940000 Sum: 1000000
Transactions:37950000 Sum: 1000000
Transactions:37960000 Sum: 1000000
Transactions:37970000 Sum: 1000000
Transactions:37980000 Sum: 1000000
Transactions:37990000 Sum: 1000000
Transactions:38000000 Sum: 1000000
Transactions:38010000 Sum: 1000000
Transactions:38020000 Sum: 1000000
Transactions:38030000 Sum: 1000000
Transactions:38040000 Sum: 1000000
Transactions:38050000 Sum: 1000000
Transactions:38060000 Sum: 1000000
Transactions:38070000 Sum: 1000000
Transactions:38080000 Sum: 1000000
Transactions:38090000 Sum: 1000000
Transactions:38100000 Sum: 1000000

```

Рисунок 1 – Результат запуску програми

Завдання 2.

Програма успішно реалізує модель "Producer-Consumer" з використанням синхронізації для забезпечення коректного доступу до спільного масиву. Тестування з різними розмірами масиву (100, 1000, 5000) показало, що програма працює правильно, без втрати або дублювання даних. Це підтверджується правильним виконанням всіх потоків і відсутністю аномалій у роботі програми.

Лістинг коду:

ProducerConsumer.java

```

package task2;

public class ProducerConsumer {
    public static void main(String[] args) {
        SharedArray sharedArray = new SharedArray(1000);
    }
}

```

```

        ProducerThread producerThread1 = new ProducerThread(sharedArray);
        ProducerThread producerThread2 = new ProducerThread(sharedArray);
        ProducerThread producerThread3 = new ProducerThread(sharedArray);
        ConsumerThread consumerThread1 = new ConsumerThread(sharedArray);
        ConsumerThread consumerThread2 = new ConsumerThread(sharedArray);

        producerThread1.start();
        producerThread2.start();
        producerThread3.start();
        consumerThread1.start();
        consumerThread2.start();

        try {
            producerThread1.join();
            producerThread2.join();
            producerThread3.join();
            consumerThread1.join();
            consumerThread2.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        System.out.println(sharedArray.array.size());
    }
}

```

ConsumerThread.java

```

package task2;

public class ConsumerThread extends Thread {
    private final SharedArray sharedArray;

    public ConsumerThread(SharedArray sharedArray) {
        this.sharedArray = sharedArray;
    }

    public void run(){
        for (int it = 0; it < 1000; it++) {
            int value = sharedArray.take();
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

```

ProducerThread.java

```

package task2;

```

```

public class ProducerThread extends Thread {
    private final SharedArray sharedArray;

    public ProducerThread(SharedArray sharedArray) {
        this.sharedArray = sharedArray;
    }

    @Override
    public void run(){
        int i = 1;
        for (int it = 0; it < 1000; it++) {
            sharedArray.put(i);
            i++;
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

```

SharedArray.java

```

package task2;

import java.util.ArrayList;

public class SharedArray {
    final ArrayList<Integer> array;
    private final int size;

    public SharedArray(int size) {
        this.size = size;
        this.array = new ArrayList<>();
    }

    public int take() {
        synchronized (this) {
            while (array.size() == 0) {
                System.out.println("EMPTY ARRAY");
                try {
                    wait();
                } catch (InterruptedException ignore) {
                }
            }
            int value = array.get(0);
            System.out.println(" CONSUMER <- " + value);
            array.remove(0);
            notifyAll();
            return value;
        }
    }
}

```



```

    }

    public void put(int element) {
        synchronized (this) {
            while (array.size() >= size) {
                System.out.println("OVERFLOWING ARRAY");
                try {
                    wait();
                } catch (InterruptedException ignore) {
                }
            }
            System.out.println(" PRODUCER -> " + element);
            array.add(element);
            notifyAll();
        }
    }
}

```

Результат:

```

PRODUCER -> 998
CONSUMER <- 663
PRODUCER -> 994
CONSUMER <- 668
PRODUCER -> 999
CONSUMER <- 667
PRODUCER -> 995
PRODUCER -> 1000
CONSUMER <- 669
CONSUMER <- 668
PRODUCER -> 996
PRODUCER -> 997
PRODUCER -> 998
PRODUCER -> 999
PRODUCER -> 1000
1000

```

Рисунок 2 – Результат запуску програми

Завдання 3.

Study.java

```

package task3;

import java.util.ArrayList;
import java.util.Arrays;

public class Study {
    public static void main(String[] args) {
        Group group1 = new Group("group1", 25);
        Group group2 = new Group("group2", 30);
        Group group3 = new Group("group3", 28);
    }
}

```

```

        Journal journal = new Journal(new
ArrayList<>(Arrays.asList(group1,group2,group3)));
        Teacher lector = new Teacher(new ArrayList<>(Arrays.asList(group1,group2,group3)),
journal);
        Teacher teacher1 = new Teacher(new ArrayList<>(Arrays.asList(group1)), journal);
        Teacher teacher2 = new Teacher(new ArrayList<>(Arrays.asList(group2)), journal);
        Teacher teacher3 = new Teacher(new ArrayList<>(Arrays.asList(group3)), journal);
        lector.start();
        teacher1.start();
        teacher2.start();
        teacher3.start();
        try {
            lector.join();
            teacher1.join();
            teacher2.join();
            teacher3.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        journal.print();
    }
}

```

Group.java

```

package task3;

import java.util.ArrayList;

public class Group {

    ArrayList<Student> students = new ArrayList<Student>();
    String name;

    Group(String name, int count) {
        for (int i = 0; i < count; i++) {
            students.add(new Student(i));
        }
        this.name = name;
    }
}

```

Journal.java

```

package task3;

import java.util.ArrayList;

public class Journal {

    ArrayList<Group> groups;
}

```

```

Journal(ArrayList<Group> groups) {
    this.groups = groups;
}

void addMark(String groupName, int studentIndex, int mark) {
    for (int groupIndex = 0; groupIndex < groups.size(); groupIndex++) {
        Group currentGroup = groups.get(groupIndex);
        if (currentGroup.name == groupName) {
            currentGroup.students.get(studentIndex).addMark(mark);
        }
    }
}

void print() {
    for (int groupIndex = 0; groupIndex < groups.size(); groupIndex++) {
        Group group = groups.get(groupIndex);
        for (int studentIndex = 0; studentIndex < group.students.size();
studentIndex++) {
            System.out.println(group.name + " " + studentIndex + " " +
group.students.get(studentIndex).marks.size());
        }
    }
}
}

```

Student.java

```

package task3;

import java.util.ArrayList;

public class Student {

    int index;
    ArrayList<Integer> marks = new ArrayList<Integer>();

    Student(int index) {
        this.index = index;
    }

    synchronized void addMark(int mark) {
        marks.add(mark);
    }
}

```

Teacher.java

```
package task3;

import java.util.ArrayList;
import java.util.Random;

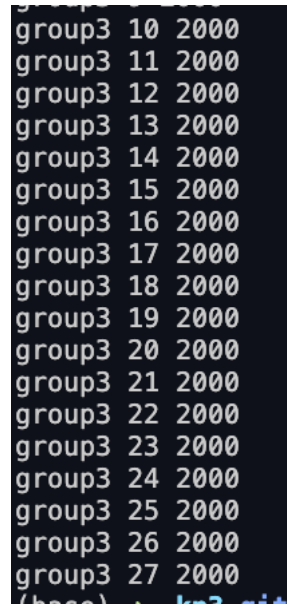
public class Teacher extends Thread {

    int WEEK_COUNT = 1000;
    Random random = new Random();

    ArrayList<Group> availableGroups;
    Journal journal;

    Teacher(ArrayList<Group> availableGroups, Journal journal) {
        this.availableGroups = availableGroups;
        this.journal = journal;
    }

    public void run() {
        for (int weekIndex = 0; weekIndex < WEEK_COUNT; weekIndex++) {
            for (int groupIndex = 0; groupIndex < availableGroups.size(); groupIndex++) {
                Group group = availableGroups.get(groupIndex);
                for (int studentIndex = 0; studentIndex < group.students.size();
studentIndex++) {
                    int mark = random.nextInt(101);
                    journal.addMark(group.name, studentIndex, mark);
                }
            }
        }
    }
}
```

Результат:

```
group3 10 2000
group3 11 2000
group3 12 2000
group3 13 2000
group3 14 2000
group3 15 2000
group3 16 2000
group3 17 2000
group3 18 2000
group3 19 2000
group3 20 2000
group3 21 2000
group3 22 2000
group3 23 2000
group3 24 2000
group3 25 2000
group3 26 2000
group3 27 2000
```

Рисунок 3 – Результат запуска програми

ВИСНОВКИ

В результаті роботи над комп'ютерним практикумом було виконано 3 завдання на мові програмування Java.

Завдання 1: Було реалізовано програму, що імітує пересилання банком грошей між акаунтами клієнтів та синхронізує їх.

Завдання 2: Було реалізовано програму, що вирішує задачу Producer-Consumer

Завдання 3: : Було реалізовано програму, що імітує виставлення викладачами оцінок студентам різних груп

Використання методів управління потоками в Java є важливою складовою для розробки багатопотокових програм. В цілому, правильне використання методів допомагає створювати ефективні та надійні багатопотокові програми, які можуть працювати ефективно на сучасних багатопроцесорних та багатопотокових системах.