МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З лабораторної роботи № 3 з дисципліни

«Технології паралельних обчислень»

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

| **Виконав(ла)** | *ІП-14 Бабіч Денис* |  |  |
| --- | --- | --- | --- |
|  | (шифр, прізвище, ім'я, по батькові) |  |  |

| **Перевірив** | *Дифучина О. Ю.* |  |  |
| --- | --- | --- | --- |
|  | (шифр, прізвище, ім'я, по батькові) |  |  |

Київ 2024

# ОСНОВНА ЧАСТИНА

**Мета роботи**: Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи.

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.



Рисунок 1.1 – Результат роботи несинхронізованого підходу



Рисунок 1.2 – Результат роботи синхронізованого підходу

Спосіб обробки фінансів не є синхронізованим, тому, якщо кілька потоків намагаються виконати транзакцію одночасно, це може призвести до проблеми Race Condition, коли баланс рахунку може стати від’ємним або загальна сума всіх рахунків може не залишатися постійною.

1. Синхронізація за допомогою локера.

**Лістинг методу transferWithLock**

public void transferWithLock(int senderIndex, int receiverIndex, int amount) {

this.lock.lock();

try {

this.accounts[senderIndex] -= amount;

this.accounts[receiverIndex] += amount;

++this.transactionsCount;

if (this.transactionsCount % NTEST == 0) {

test();

}

}

finally {

this.lock.unlock();

}

}

1. Синхронізація за допомогою синхронізованого блока.

**Лістинг методу transferWithSynchronizedBlock**

public void transferWithSynchronizedBlock(int senderIndex, int receiverIndex, int amount) {

synchronized(this)

{

this.accounts[senderIndex] -= amount;

this.accounts[receiverIndex] += amount;

++this.transactionsCount;

if (this.transactionsCount % NTEST == 0) {

test();

}

}

}

1. Синхронізація за допомогою синхронізованого методу.

**Лістинг методу transferWithSynchronizedMethod**

public synchronized void transferWithSynchronizedMethod(int senderIndex, int receiverIndex, int amount)

{

this.accounts[senderIndex] -= amount;

this.accounts[receiverIndex] += amount;

++this.transactionsCount;

if (this.transactionsCount % NTEST == 0)

{

test();

}

}

1. Реалізуйте приклад Producer-Consumer application. 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.

**Лістинг класу Producer**

final class Producer implements Runnable {

public static final int INTERRUPT\_VALUE = -1;

private Drop drop;

private int[] dataBuffer;

public Producer(Drop drop, int dataBufferSize) {

this.drop = drop;

this.dataBuffer = new int[dataBufferSize];

}

public void run() {

Random random = new Random();

for (int i = 0; i < this.dataBuffer.length; ++i) {

this.dataBuffer[i] = i;

this.drop.put(this.dataBuffer[i]);

try {

Thread.sleep(random.nextInt(5000));

} catch (InterruptedException e) {

System.out.println(e.getStackTrace());

}

}

this.drop.put(Producer.INTERRUPT\_VALUE);

}

}

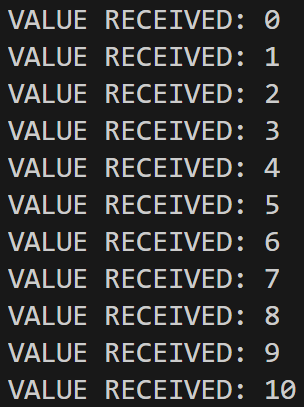


Рисунок 1.3 – Результат виконання програми

1. 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.

**Лістинг класу Group**

final class Group {

private static int idCounter = 0;

private final int ID;

private final String NAME;

private final ArrayList<Student> STUDENTS;

public Group(String name, Student... students) {

this.ID = ++Group.idCounter;

this.NAME = name;

this.STUDENTS = new ArrayList<>(Arrays.asList(students));

}

public static int getGroupsCount() {

return Group.idCounter;

}

public long getId() {

return this.ID;

}

public String getName() {

return this.NAME;

}

public int getStudentsCount() {

return this.STUDENTS.size();

}

public Student getStudent(int index) {

if (index >= this.STUDENTS.size() || index < 0) {

throw new IllegalArgumentException("Sequence number is out of bounds.");

}

return this.STUDENTS.get(index);

}

public int getStudentIndex(Student student) {

if (student == null) {

throw new IllegalArgumentException("Student cannot be null");

}

final int RESULT\_FAILURE = -1;

int index = this.STUDENTS.indexOf(student);

return (index != RESULT\_FAILURE) ? index : RESULT\_FAILURE;

}

public boolean containsStudent(Student student) {

if (student == null) {

throw new IllegalArgumentException("Student cannot be null");

}

return this.STUDENTS.contains(student);

}

}

**Лістинг класу Journal**

final class Journal

{

public static final int GRADE\_MIN\_VALUE = 0;

public static final int GRADE\_MAX\_VALUE = 100;

private static int idCounter = 0;

private final int ID;

private final String NAME;

private final int WEEKS\_COUNT;

private final ConcurrentHashMap<Student, ReentrantLock[]> SHEET\_LOCKS;

private final HashMap<Group, ConcurrentHashMap<Student, Integer[]>> SHEET;

public Journal(String name, int weeksCount, Group... groups)

{

this.ID = ++Journal.idCounter;

this.NAME = name;

this.WEEKS\_COUNT = weeksCount;

this.SHEET = new HashMap<>();

this.SHEET\_LOCKS = new ConcurrentHashMap<>();

for (Group group : groups)

{

for (int i = 0; i < group.getStudentsCount(); ++i)

{

this.SHEET\_LOCKS.computeIfAbsent(group.getStudent(i), k -> IntStream.range(0, weeksCount).mapToObj(j -> new ReentrantLock()).toArray(ReentrantLock[]::new));

this.SHEET.computeIfAbsent(group, k -> new ConcurrentHashMap<>()).put(group.getStudent(i), IntStream.range(0, weeksCount).mapToObj(j -> 0).toArray(Integer[]::new));

}

}

}

public int getId()

{

return this.ID;

}

public String getName()

{

return this.NAME;

}

public int getWeeksCount()

{

return this.WEEKS\_COUNT;

}

public void putGrade(Student student, int weekIndex, int grade)

{

if (student == null)

{

throw new IllegalArgumentException("Student cannot be null");

}

if (weekIndex < 0 || weekIndex >= this.WEEKS\_COUNT)

{

throw new IllegalArgumentException("Invalid weekIndex value.");

}

if (grade < Journal.GRADE\_MIN\_VALUE || grade > Journal.GRADE\_MAX\_VALUE)

{

throw new IllegalArgumentException("Grade value is invalid.");

}

ReentrantLock lock = this.SHEET\_LOCKS.get(student)[weekIndex];

lock.lock();

try

{

for (Group group : this.SHEET.keySet())

{

if (group.containsStudent(student)) {

this.SHEET.get(group).get(student)[weekIndex] += grade;

break;

}

}

} finally {

lock.unlock();

}

}

public void printSheet() {

List<Group> sortedGroups = new ArrayList<>(this.SHEET.keySet()).stream().sorted(Comparator.comparing(Group::getName)).collect(Collectors.toList());

for (Group group : sortedGroups) {

System.out.println(group.getName());

List<Student> sortedStudents = new ArrayList<>(this.SHEET.get(group).keySet()).stream().sorted(Comparator.comparing(Student::getFullName)).collect(Collectors.toList());

for (Student student : sortedStudents) {

System.out.printf("%-10s: ", student.getFullName());

for (int j = 0; j < this.WEEKS\_COUNT; ++j) {

System.out.printf("%-3d|", this.SHEET.get(group).get(student)[j]);

}

System.out.println();

}

System.out.println();

}

}

}

**Лістинг класу Student**

package task3;

final class Student

{

private static int idCounter = 0;

private final int ID;

private String name;

private String surname;

private String patronymic;

private String fullName;

public Student(String name, String surname, String patronymic)

{

this.ID = ++Student.idCounter;

this.name = name;

this.surname = surname;

this.patronymic = patronymic;

this.fullName = String.format("%s %s %s", surname, name, patronymic);

}

public static int getStudentsCount()

{

return Student.idCounter;

}

public long getId()

{

return this.ID;

}

public String getName()

{

return this.name;

}

public String getSurname()

{

return this.surname;

}

public String getPatronymic()

{

return this.patronymic;

}

public String getFullName()

{

return this.fullName;

}

}

**Лістинг класу Teacher**

final class Teacher {

private static int idCounter = 0;

private final int ID;

private String name;

private String surname;

private String patronymic;

private String fullName;

public Teacher(String name, String surname, String patronymic) {

this.ID = ++Teacher.idCounter;

this.name = name;

this.surname = surname;

this.patronymic = patronymic;

this.fullName = String.format("%s %s %s", surname, name, patronymic);

}

public static int getAcademicsCount() {

return Teacher.idCounter;

}

public long getId() {

return this.ID;

}

public String getName() {

return this.name;

}

public String getSurname() {

return this.surname;

}

public String getPatronymic() {

return this.patronymic;

}

public String getFullName() {

return this.fullName;

}

public int generateGrade(int min, int max) {

if (min > max)

{

throw new IllegalArgumentException("min must be less than or equal to max");

}

return min + (int)(Math.random() \* ((max - min) + 1));

}

}

**Лістинг класу Main**

package task3;

import java.util.ArrayList;

final class Main

{

public static void main(String[] args)

{

Group group1 = new Group("Group 1", Main.generateStudentsArray(25));

Group group2 = new Group("Group 2", Main.generateStudentsArray(30));

Group group3 = new Group("Group 3", Main.generateStudentsArray(27));

Journal journal = new Journal("Journal", 10, group1, group2, group3);

Teacher lecturer = new Teacher("Lecturer", "Lecturer", "Lecturer");

Teacher assistant1 = new Teacher("Assistant 1", "Assistant 1", "Assistant 1");

Teacher assistant2 = new Teacher("Assistant 2", "Assistant 2", "Assistant 2");

Teacher assistant3 = new Teacher("Assistant 3", "Assistant 3", "Assistant 3");

Group[] groups = { group1, group2, group3 };

Teacher[] teachers = { lecturer, assistant1, assistant2, assistant3 };

ArrayList<Thread> threads = new ArrayList<>();

for (Teacher teacher : teachers)

{

threads.addLast(new Thread(() ->

{

for (Group group : groups)

{

for (int i = 0; i < group.getStudentsCount(); ++i)

{

for (int weekIndex = 0; weekIndex < journal.getWeeksCount(); ++weekIndex)

{

journal.putGrade(group.getStudent(i), weekIndex, teacher.generateGrade(Journal.GRADE\_MIN\_VALUE, Journal.GRADE\_MAX\_VALUE));

}

}

}

}));

}

for (Thread thread : threads)

{

thread.start();

}

for (Thread thread : threads)

{

try

{

thread.join();

}

catch (InterruptedException ex)

{

System.out.println(ex.getStackTrace());

}

}

journal.printSheet();

}

private static Student[] generateStudentsArray(int count)

{

Student[] students = new Student[count];

String placeholder;

for (int i = 0; i < count; ++i)

{

placeholder = Integer.toString(i);

students[i] = new Student(placeholder, placeholder, placeholder);

}

return students;

}

}

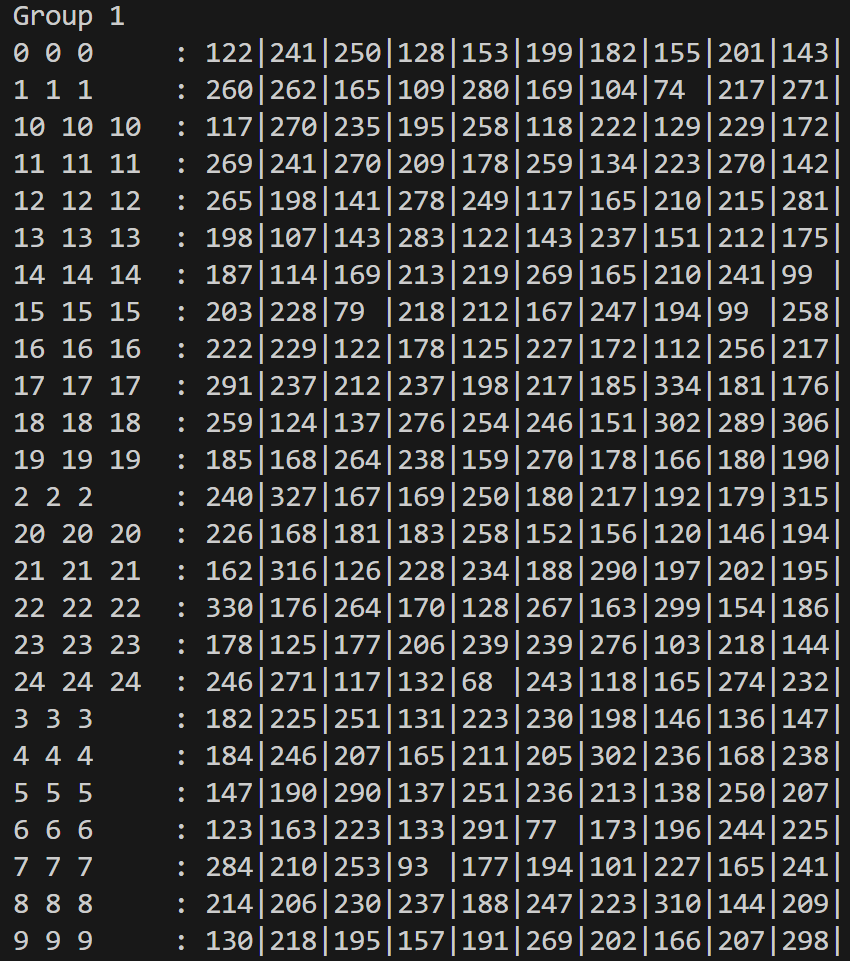


Рисунок 1.4 – Отримані оцінки для першої групи

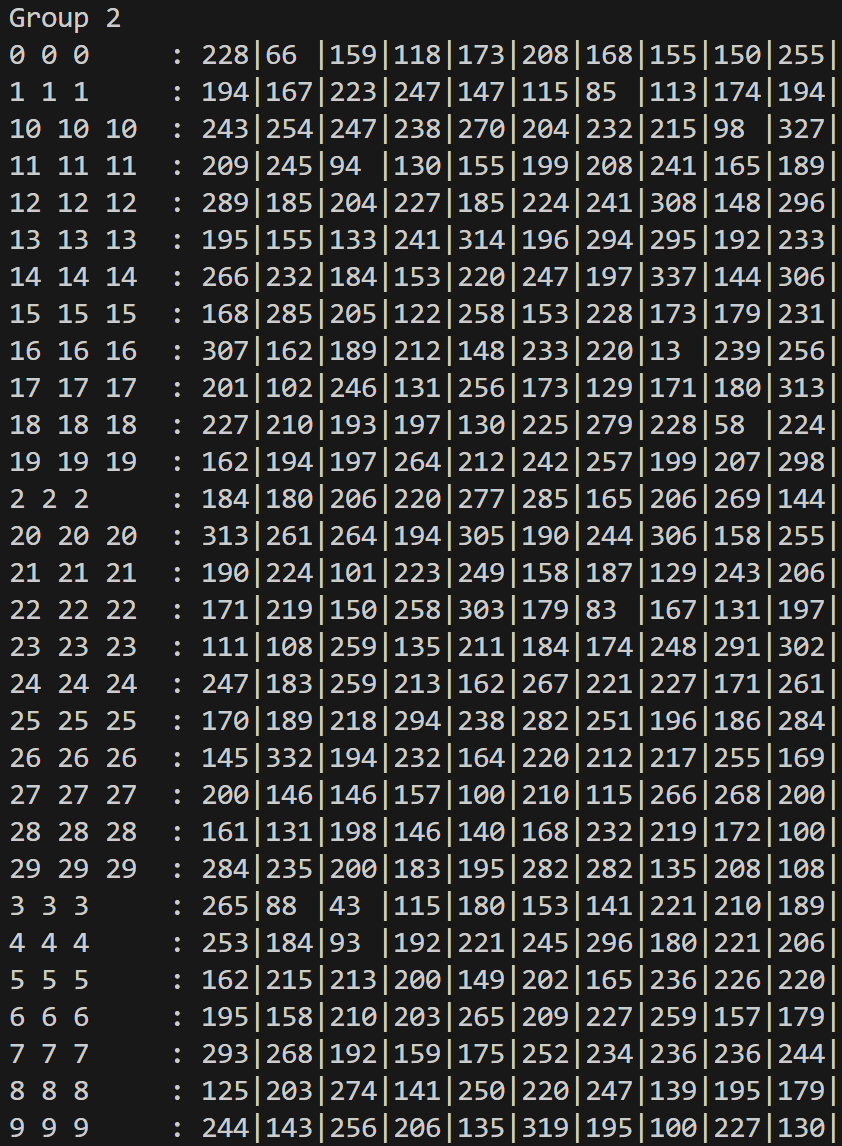


Рисунок 1.5 – Отримані оцінки для другої групи

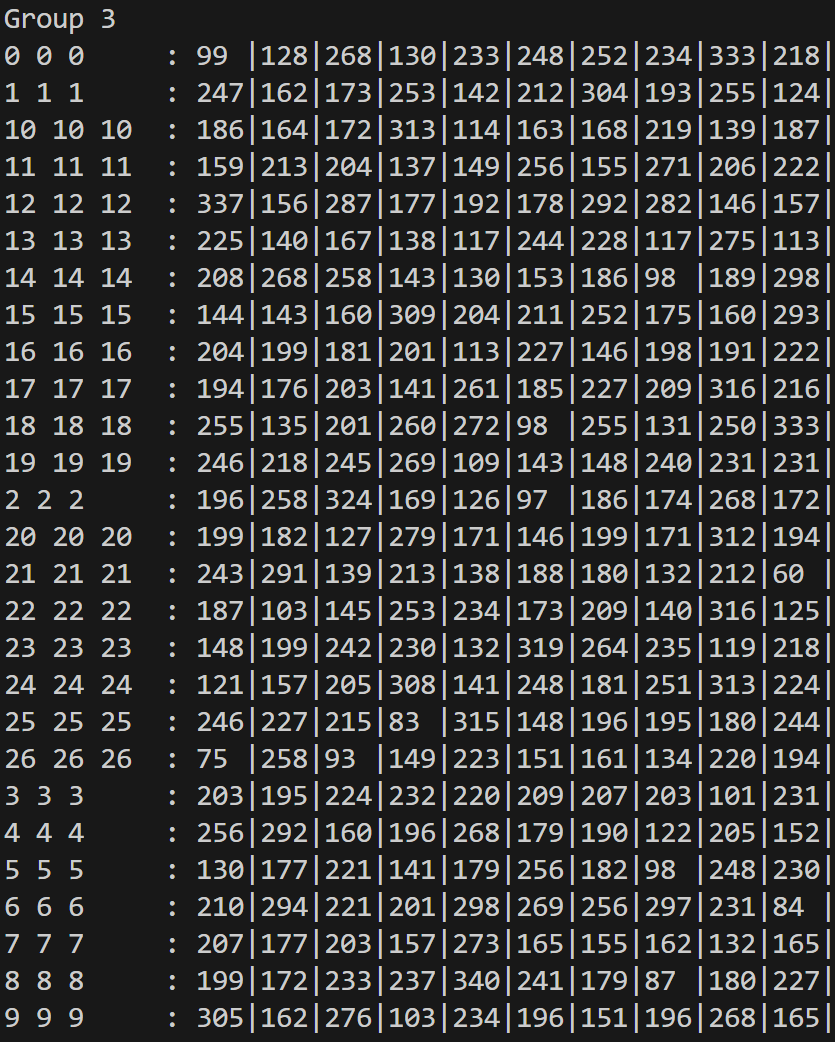


Рисунок 1.6 – Отримані оцінки для третьої групи

# ВИСНОВКИ

Під час виконання даної роботи було розглянуто та реалізовано різні методи управління потоками на прикладі різних завдань у Java. Було використано синхронізацію, блокування для забезпечення коректної роботи програм. В першому завданні було реалізовано програмний код з різними методами управління потоками, що дозволило забезпечити коректну роботу програми симуляції банкової системи. У другому завданні було реалізовано приклад Producer-Consumer application, де було модифіковано масив даних, який передається між відправником та споживачем. Третє завдання вимагало реалізації електронного журналу групи. Завдяки використанню потоків, було можливо ефективно виставляти оцінки студентам відразу багатьом викладачам. Таким чином, використання потоків допомагає забезпечити коректність роботи програм, оптимізацію використання ресурсів системи та покращують продуктивність.