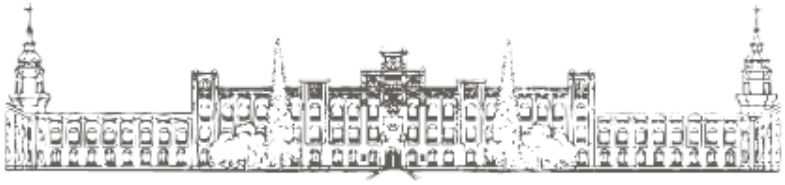

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Звіт

до лабораторного практикуму 3

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

з дисципліни

«Технології паралельних обчислень»

Виконав:
студент групи ІП-01
Пашковський Євгеній

Київ – 2023

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

Хід виконання

Виправлений код першого завдання (до методів додано синхронізацію):

```
public synchronized void transferSync(int from, int to, int amount)
{
    accounts[from] -= amount;
    accounts[to] += amount;
    ntransacts++;
    if (ntransacts % NTEST == 0) test();
}

public void transferSyncBlock(int from, int to, int amount) {
    synchronized (this) {
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0) test();
    }
}

public void transferLock(int from, int to, int amount) {
    locker.lock();
    try {
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0) test();
    } finally {
        locker.unlock();
    }
}

public synchronized void waitTransfer(int from, int to, int amount)
{
    while (accounts[from] < amount) {
        try {
            wait();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    accounts[from] -= amount;
    accounts[to] += amount;
    ntransacts++;
    notifyAll();
    if (ntransacts % NTEST == 0) test();
}
```

Реалізація Producer-Consumer:

Main:

```java
package producer_consumer;

public class Main {
    public static void main(String[] args) {
//        SharedArrayBlockingQueue<Integer> sharedBuffer = new
SharedArrayBlockingQueue<>(1000);
        SharedCustomBuffer<Integer> sharedBuffer = new
SharedCustomBuffer<>(1000);

        ProducerThread producerThread = new
ProducerThread(sharedBuffer);
        ConsumerThread consumerThread = new
ConsumerThread(sharedBuffer);

        producerThread.start();
        consumerThread.start();
        try {
            producerThread.join();
            consumerThread.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

SharedBuffer:

```java
package producer_consumer;

public interface SharedBuffer<T> {
    T take() throws InterruptedException;

    void put(T object) throws InterruptedException;

    int getMaximumSize();
}
```

SharedArrayBlockingQueue:

```java
package producer_consumer;

import java.util.concurrent.ArrayBlockingQueue;

public class SharedArrayBlockingQueue<T> implements SharedBuffer<T>
{

    private final ArrayBlockingQueue<T> queue;
    private int maximumSize = 0;

    public SharedArrayBlockingQueue(int capacity) {
        this.queue = new ArrayBlockingQueue<>(capacity);
    }

    public T take() throws InterruptedException {
        return queue.take();
    }
```

```
    public void put(T object) throws InterruptedException {
        queue.put(object);
        synchronized (this) {
            int size = queue.size();
            if (size > maximumSize) maximumSize = size;
        }
    }

    public synchronized int getMaximumSize() {
        return this.maximumSize;
    }
}
```

SharedCustomBuffer:

```
package producer_consumer;

import java.util.concurrent.ArrayBlockingQueue;

public class SharedArrayBlockingQueue<T> implements SharedBuffer<T>
{

    private final ArrayBlockingQueue<T> queue;
    private int maximumSize = 0;

    public SharedArrayBlockingQueue(int capacity) {
        this.queue = new ArrayBlockingQueue<>(capacity);
    }

    public T take() throws InterruptedException {
        return queue.take();
    }

    public void put(T object) throws InterruptedException {
        queue.put(object);
        synchronized (this) {
            int size = queue.size();
            if (size > maximumSize) maximumSize = size;
        }
    }

    public synchronized int getMaximumSize() {
        return this.maximumSize;
    }
}
```

ProducerThread:

```
package producer_consumer;

import java.text.SimpleDateFormat;
import java.util.Date;

public class ProducerThread extends Thread {
    private final SharedBuffer<Integer> sharedBuffer;

    private final int MAX_VALUE = 100000;
```

```
    public ProducerThread(SharedBuffer<Integer> sharedBuffer) {
        this.sharedBuffer = sharedBuffer;
    }

    @Override
    public void run() {
        while (true) {
            try {
                int value =
Math.toIntExact(Math.round(Math.random() * MAX_VALUE));
                sharedBuffer.put(value);
                SimpleDateFormat sdfDate = new
SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
                Date now = new Date();
                System.out.println("[" + sdfDate.format(now) + "]"
+ " [PRODUCER] Sent message: " + value + ". Shared buffer maximum
size: " + sharedBuffer.getMaximumSize());
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

ConsumerThread:

```
package producer_consumer;

import java.text.SimpleDateFormat;
import java.util.Date;

public class ConsumerThread extends Thread {

    private final SharedBuffer<Integer> sharedBuffer;

    public ConsumerThread(SharedBuffer<Integer> sharedBuffer) {
        this.sharedBuffer = sharedBuffer;
    }

    @Override
    public void run() {
        while (true) {
            try {
                int value = sharedBuffer.take();
                SimpleDateFormat sdfDate = new
SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
                Date now = new Date();
                System.out.println("[" + sdfDate.format(now) + "]"
+ " [CONSUMER] Got message: " + value);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Алгоритм для заповнення журналу:

Main:

```
package journal;

public class Main {
    public static void main(String[] args) {
        Group group1 = Group.randomGroup(50);
        Group group2 = Group.randomGroup(30);
        Group group3 = Group.randomGroup(40);

        Journal journal = new Journal();

        journal.addGroup(group1);
        journal.addGroup(group2);
        journal.addGroup(group3);

        Lecturer lecturer = new Lecturer(journal);
        Assistant assistant1 = new Assistant(journal, group1);
        Assistant assistant2 = new Assistant(journal, group2);
        Assistant assistant3 = new Assistant(journal, group3);

        for (int i = 0; i < 10; i++) {
            try {
                lecturer.addGrades();
                assistant1.addGrades();
                assistant2.addGrades();
                assistant3.addGrades();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }


System.out.println(journal.getGrades(group1.getStudentsList().get(0
)));
    }
}
```

Student:

```
package journal;

import java.util.UUID;

public class Student {
    private final UUID id = UUID.randomUUID();

    private Group group;

    public String getId() {
        return id.toString();
    }
}
```

Lecturer:

```
package journal;
```

```java
public class Lecturer {
    private final Journal journal;

    public Lecturer(Journal journal) {
        this.journal = journal;
    }

    public void addGrades() throws InterruptedException {
        GradeThread gradeThread = new GradeThread(journal, null);

        gradeThread.start();
        gradeThread.join();
    }
}
```

Journal:

```java
package journal;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Journal {
    private final Map<Student, List<Integer>> studentGradesMap =
new HashMap<>();

    private final List<Group> groupList = new ArrayList<>();

    public synchronized void addGroup(Group group) {
        groupList.add(group);

        for (Student student : group.getStudentsList()) {
            if (!studentGradesMap.containsKey(student))
                studentGradesMap.put(student, new ArrayList<>());
        }
    }

    public void addGrade(Student student, int grade) {
        synchronized (studentGradesMap) {
            if (grade < 0 || grade > 100) throw new
RuntimeException("Wrong grade");

            studentGradesMap.get(student).add(grade);
        }
    }

    public List<Group> getGroupList() {
        synchronized (groupList) {
            return new ArrayList<>(groupList);
        }
    }

    public List<Integer> getGrades(Student student) {
        synchronized(studentGradesMap) {
            return this.studentGradesMap.get(student);
```

```
        }
    }
}
```

Group:

```
package journal;

import java.util.ArrayList;
import java.util.List;

public class Group {
    private final List<Student> studentsList = new ArrayList<>();

    public void addStudent(Student student) {
        if (studentsList.contains(student)) throw new
RuntimeException("Student is already in this group");

        studentsList.add(student);
    }

    public void deleteStudent(Student student) {
        if (!studentsList.contains(student)) throw new
RuntimeException("Student doesn't exists in this group");

        studentsList.remove(student);
    }

    public static Group randomGroup(int capacity) {
        Group group = new Group();
        for (int i = 0; i < capacity; i++) {
            group.addStudent(new Student());
        }

        return group;
    }

    public List<Student> getStudentsList() {
        return new ArrayList<>(studentsList);
    }
}
```

GradeThread:

```
package journal;

import java.util.ArrayList;
import java.util.List;

public class GradeThread extends Thread {
    private final Journal journal;
    private final Group specifiedGroup;

    public GradeThread(Journal journal, Group specifiedGroup) {
        this.journal = journal;
        this.specifiedGroup = specifiedGroup;
    }
```

```
    @Override
    public void run() {
        List<Group> groupList = journal.getGroupList();

        if (specifiedGroup != null) {
            groupList = new ArrayList<>();
            groupList.add(specifiedGroup);
        };

        for (Group group : groupList) {
            List<Student> studentList = group.getStudentsList();

            for (Student student : studentList) {
                journal.addGrade(student,
Math.toIntExact(Math.round(Math.random() * 100)));
            }
        }
    }
}
```

Assistant:

```
package journal;

public class Assistant {
    private final Journal journal;

    private final Group specifiedGroup;

    public Assistant(Journal journal, Group specifiedGroup) {
        this.journal = journal;
        this.specifiedGroup = specifiedGroup;
    }

    public void addGrades() throws InterruptedException {
        GradeThread gradeThread = new GradeThread(journal,
specifiedGroup);

        gradeThread.start();
        gradeThread.join();
    }
}
```

Висновки

Виконуючи роботу комп’ютерного практикуму, було виправлено наданий код, використавши синхронізацію, реалізовано алгоритм для вирішення задачі Producer-Consumer а також створено алгоритм для заповнення журналу одночасно декількома викладачами за допомогою паралельних обчислень. Ознайомився з методикою та необхідністю синхронізації та з помилками, що можуть виникнути під час паралельних обчислень.