НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіти до комп’ютерних практикумів з кредитного модуля

“Parallel computing”

**Виконав**

**Студенти групи ІТ-02**

**Макаров І.С.**

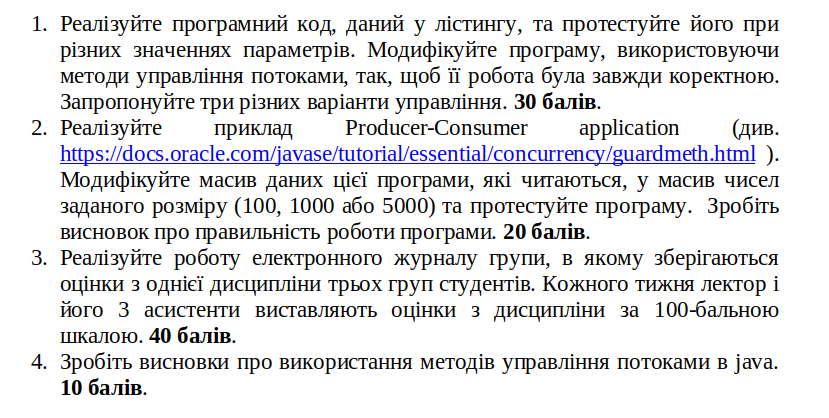
**Перевірив:**

Київ – 2022

**Комп‘ютерний практикум No 3**

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

ЗАВДАННЯ



**ВИКОНАННЯ**

Повний код лаби: *https://github.com/kinfi4/tmp/tree/PARALEL-LAB-3*

Я залишу тут лише базовий код, повний можете подивитись на гіті, та і я буду захищати цю роботу, там покажу, та розпишу детальніше.

**TASK1**

Methods for money transfer. Three types of synchronization:

public synchronized void syncTransfer(int from, int to, int amount) throws InterruptedException {

accounts[from] -= amount;

accounts[to] += amount;

ntransacts++;

if (ntransacts % NTEST == 0) {

test();

}

}

public synchronized void syncBlockTransfer(int from, int to, int amount) throws InterruptedException {

synchronized (sync) {

accounts[from] -= amount;

accounts[to] += amount;

ntransacts++;

if (ntransacts % NTEST == 0) {

test();

}

}

}

public synchronized void lockTransfer(int from, int to, int amount) throws InterruptedException {

this.lock.lock();

try {

accounts[from] -= amount;

accounts[to] += amount;

ntransacts++;

if (ntransacts % NTEST == 0) {

test();

}

} finally {

this.lock.unlock();

}

}

**TASK2**

**Buffer:**

**public class Buffer {**

**private final int[] items = new int[20];**

**private int currentPutIdx = 0;**

**private int currentTakeIdx = 0;**

**private int itemsCount = 0;**

**public synchronized int take() {**

**while (itemsCount == 0) {**

**try {**

**wait();**

**} catch (InterruptedException e) {}**

**}**

**int item = this.items[currentTakeIdx];**

**this.currentTakeIdx += 1;**

**if(this.currentTakeIdx == this.items.length) {**

**this.currentTakeIdx = 0;**

**}**

**this.itemsCount -= 1;**

**notifyAll();**

**return item;**

**}**

**public synchronized void put(int item) {**

**while (itemsCount == this.items.length) {**

**try {**

**wait();**

**} catch (InterruptedException e) {}**

**}**

**this.items[this.currentPutIdx] = item;**

**this.currentPutIdx += 1;**

**if(this.currentPutIdx == this.items.length) {**

**this.currentPutIdx = 0;**

**}**

**this.itemsCount += 1;**

**notifyAll();**

**}**

**}**

**Producer:**

public void run() {

int[] range = IntStream.*iterate*(1, n -> n + 1).limit(1000).toArray();

Random random = new Random();

for (int i : range) {

drop.put(i);

try {

Thread.*sleep*(random.nextInt(50));

} catch (InterruptedException ignored) {}

}

drop.put(-1);

}

**Consumer:**

public void run() {

Random random = new Random();

int message = drop.take();

while(message != -1) {

try {

System.*out*.println("MESSAGE RECEIVED: " + message);

message = drop.take();

Thread.*sleep*(random.nextInt(50));

} catch (InterruptedException ignored) {}

}

}

**TASK3**

Grades book class:

public synchronized boolean allMarksPlaced() {

for (int[] group : this.gradesBook) {

for (int mark : group) {

if (mark == 0) {

return false;

}

}

}

return true;

}

public int[] getGroup(int groupId) {

synchronized (this) {

while(this.blockedGroups.contains(groupId)) {

try {

wait();

} catch (InterruptedException ignored) {}

}

this.blockedGroups.add(groupId);

return this.gradesBook[groupId];

}

}

public void unBlockGroup(int groupId) {

synchronized (this) {

this.blockedGroups.remove((Integer) groupId);

notifyAll();

}

}

Teacher class:

@Override

public void run() {

Random rand = new Random();

while(!this.book.allMarksPlaced()) {

int randomGroup = rand.nextInt(0, 3);

int randomStudent = rand.nextInt(0, *STUDENTS\_PER\_GROUP*);

int[] group = this.book.getGroup(randomGroup);

if(group[randomStudent] == 0) {

int randomMark = rand.nextInt(1, 101);

group[randomStudent] = randomMark;

}

this.book.unBlockGroup(randomGroup);

try {

Thread.*sleep*(10);

} catch (InterruptedException ignored) {}

}

}

Main method:

public static void task3() {

GradesBook book = new GradesBook();

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

for (int i = 0; i < 4; i++) {

Thread teacher = new Teacher(book);

threads.add(teacher);

teacher.start();

}

for (Thread tr: threads) {

try {

tr.join();

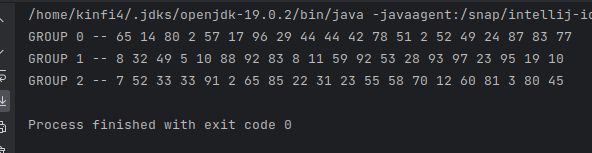
} catch (InterruptedException ignored) {}

}

book.printMarks();

}

Result:



**ВИСНОВОК**

В рамках работи, ми на практиці познайомились с механізмами синхронізації потоків в java. Серед яких: synchronized метод, synchronized блок, Lockers ітд. Окрім того, в останньому завданні, я писав можна сказати кастомну синхронізацію з блокуванням оцінок певної групи, якщо інший поток вже щось з цією групою робить.

Що тут можна додати, звісно дана тема є неймовірно важливою для ТПО, адже це основа без якої не можливо написати жодний алгоритм, де є якийсь спільний для декількох потоків ресурс та не атомарні операції з даним ресурсом.