Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум №3**

з дисципліни «Технології паралельних обчислень»

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

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІТ-04  Коновальчук Андрій Володимирович  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила**:  асистент кафедри ІПІ  Дифучина Олександра Юріївна |

Київ 2023

**Хід роботи:**

***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.

*Опис пункту:*

Даний лістинг коду містить 3 класи: AsyncBankTest, Bank, TransferThread.

AsyncBankTest містить фіксовану n кількість акаунтів та фіксоване значення початкового балансу цих акаунтів. Тут же, в методі main ми створюємо об’єкт класу Bank та створюємо та запускаємо в циклі n кількість потоків з різницею в пріоритеті ±1.

Потоки створені об’єктом класу TransferThread, котрий розширює клас Thread, і приймає такі значення: об’єкт класу Bank, індекс акаунуту та початковий баланс акаунту. В методі run цього класу виконується встановлена кількість транзакцій для даного акаунту, де пересилається випадкова сума випадковому акаунту.

В класі Bank є два важливих методи – transfer і test. Метод transfer виконує пересилку даної кількості кошті з одного акаунта в інший, інкрементує значення поточної транзакції в банку та робить перевірку чи число транзакцій не кратне фіксованому значенню n\_test. Якщо воно кратне, то виконується метод test, котрий вказує кількість транзакцій та суму коштів з усіх акаунтів.

*Приклад коду:*

Для виконання задачі і правильної роботи коду потрібно написати додаткові методи, з використанням методів управління потоками. Тож було написано наступні варіанти методу transfer: синхронізований метод, з використанням методів wait та notifyAll і метод з використаннам об’єкта класу ReentrantLock.

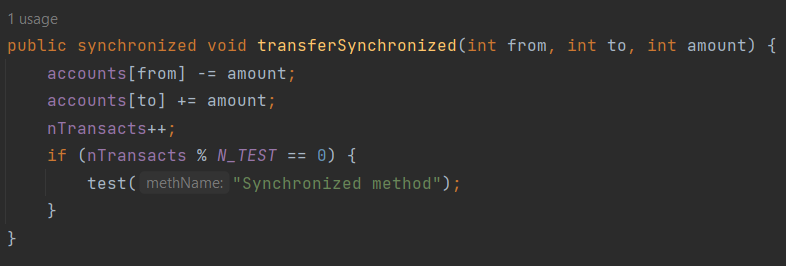


Рисунок 1 – Метод Synchronized

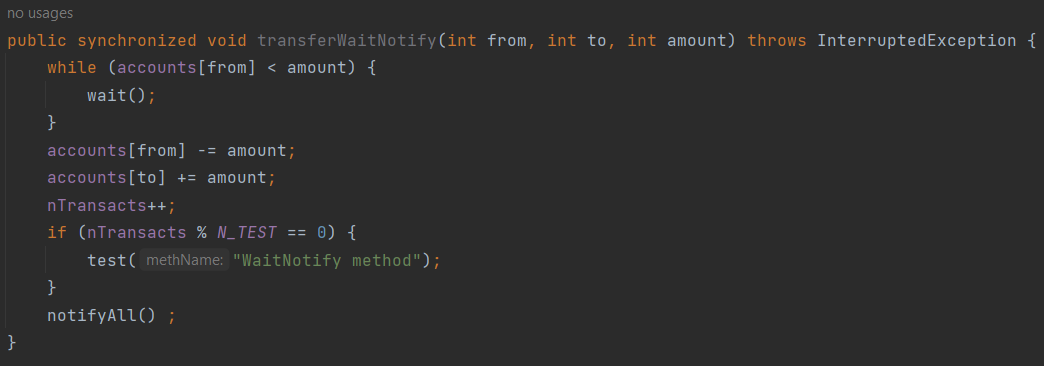


Рисунок 2 – Метод WaitNotify

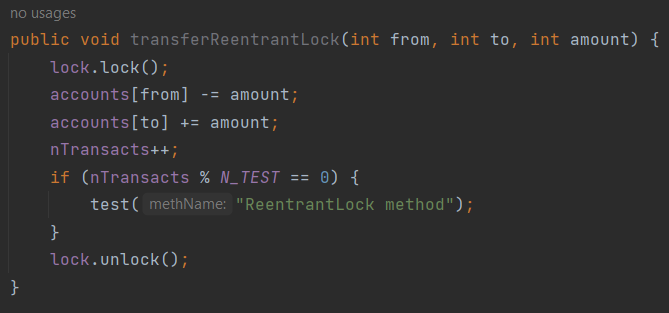


Рисунок 3 – Метод ReentrantLock

*Спостереження, результати, висновки:*

З використанням цих методів бачимо наступні результати в консолі: кількість транзакцій кратна 10000, а сума завжди однакова. Отже, код працює правильно з використанням методів управлінням потоків, задача виконана.

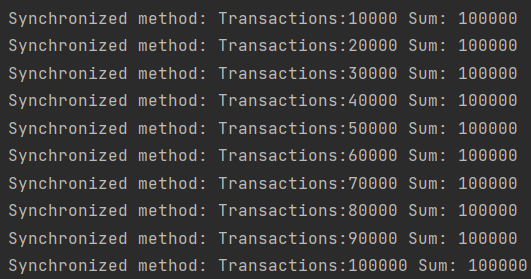


Рисунок 4 – Результати роботи методу Synchronized

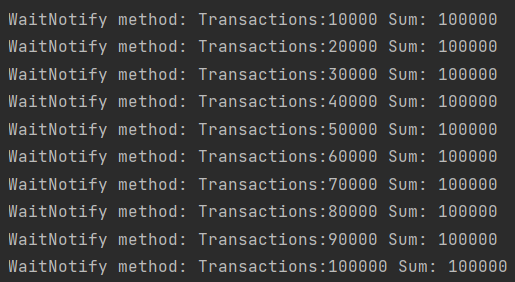


Рисунок 5 – Результати роботи методу WaitNotify

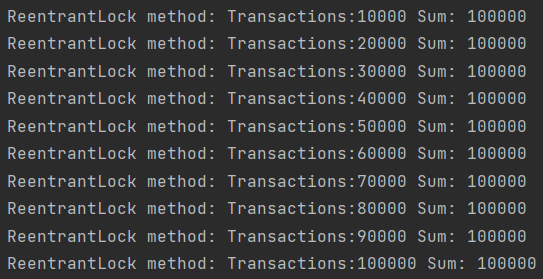


Рисунок 6 – Результати роботи методу ReentrantLock

***Завдання 2:*** Реалізуйте приклад Producer-Consumer application (див. https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html ). 12 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.

*Опис пункту:*

Задача вимагає використання Guarded Blocks та імплементації патерну Producer-Consumer. Цей вид програми обмінюється даними між двома потоками: виробником (Producer), який створює дані, і споживачем (Consumer), який щось з ними робить. Два потоки спілкуються за допомогою спільного об’єкта (Drop). Координація має важливе значення: потік-споживач не повинен намагатися отримати дані до того, як потік-виробник їх доставить, а потік-виробник не повинен намагатися доставити нові дані, якщо споживач не отримав старі дані.

У нашому варіанті дані є набором чисел (100, 1000 чи 5000), які передаються через об’єкт типу Drop. Drop є спільним об’єктом для обох потоків, і є способом передачі повідомлень між ними. Він має два методи take – для споживача і put – для виробника. Програма синхронізована за допомогою ключового слова synchronized біля цих методів та тому що це спільний об’єкт.

Потік виробника, визначений у Producer, надсилає набір чисел. Число 0 означає, що всі повідомлення надіслано. Щоб імітувати непередбачувану природу додатків реального світу, потік виробника призупиняється на випадкові проміжки часу між повідомленнями.

Потік споживача, визначений у Consumer, просто отримує повідомлення та друкує їх, доки не отримає число 0. Цей потік також призупиняється на випадкові проміжки часу.

*Спостереження, результати, висновки:*

В консолі отримуємо послідовний вивід набору чисел з випадковою затримкою. Це означає, що задача виконана, і програма працює правильно, адже споживач не отримав дані до того, як виробник їх доставив, а виробник не намагатися доставити нові дані, доки споживач не отримав старі дані



Рисунок 7 – Результати роботи Producer-Consumer

***Завдання 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.

*Опис пункту:*

Наша реалізація містить 6 класів: Mark, Student, Group, Journal, TeacherThread, Main. Логічно що перші 4 класи складають архітектуру електронного журналу, і виділити можна лише метод setMark класу Student, котрий пов’язаний з роботою потоків. Потоками є викладачі (лектор і три асистенти), котрі виставляють оцінки, використовуючи метод setMark, котрий контролюється методами об’єкта класу ReentrantLock (нам зручно використати саме його).

*Приклад коду:*

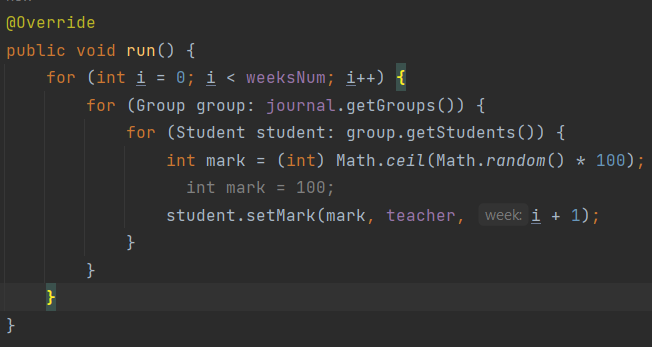


Рисунок 8 – Метод run класу TeacherThread

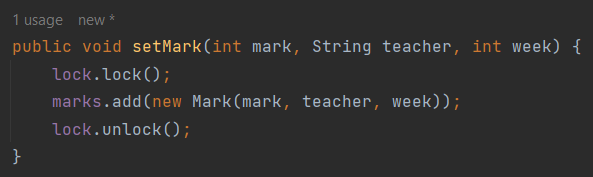


Рисунок 9 – Метод setMark класу Student

*Результати роботи:*

При виводі оцінок з журналу бачимо візуалізацію роботи потоків: одному студенту за один тиждень виставлено оцінку чотирма викладачами; дивлячись на стовпчики, бачимо “зміщення” між ними по викладачах. Зафіксувавши значення балу, бачимо в сумі однакове число у кожного студента, отже, бали виставлені у кожного, і програма працює справно.

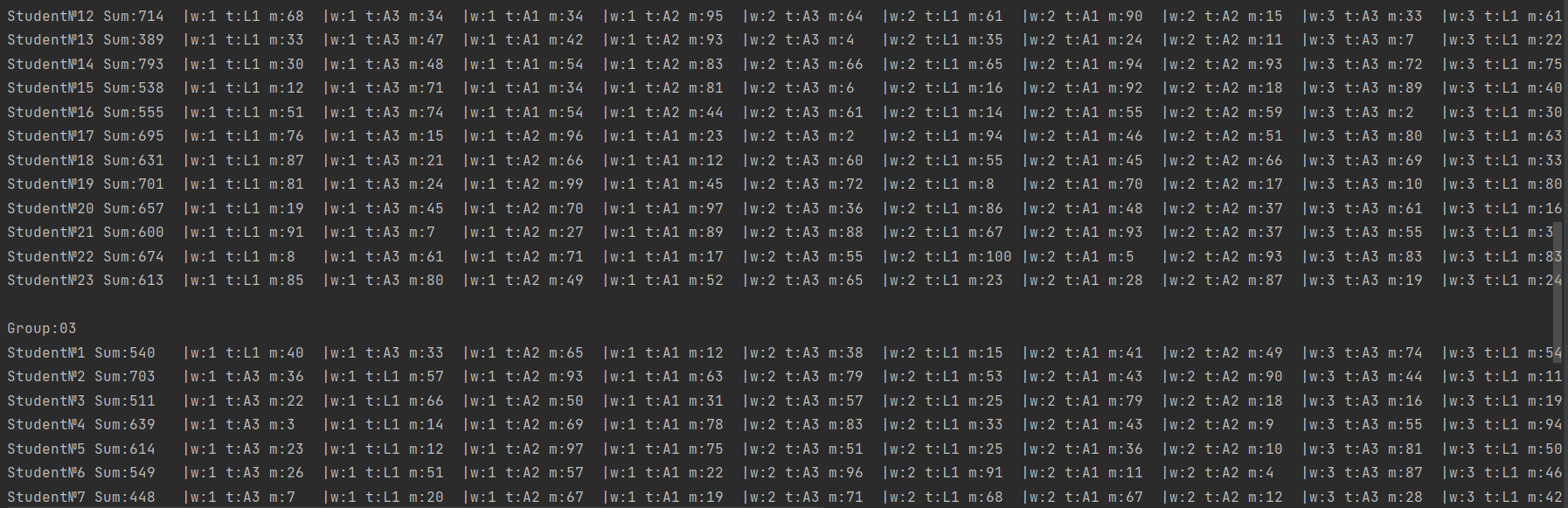


Рисунок 8 – Вівід журналу

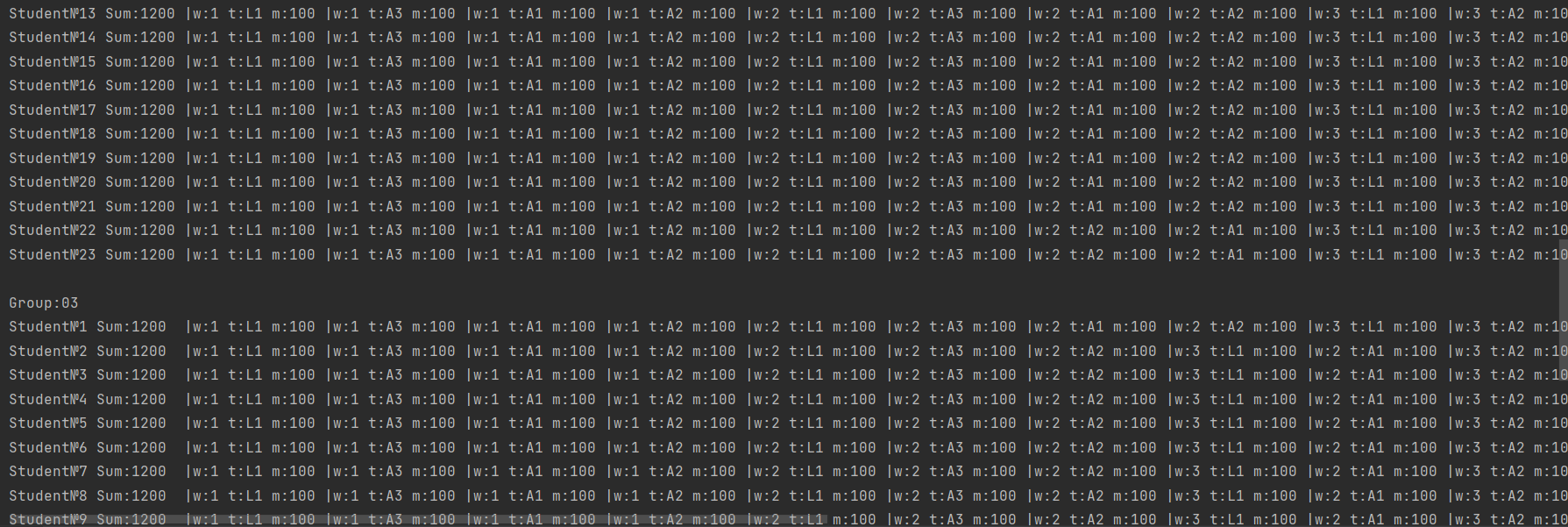


Рисунок 9 – Журнал з максимальними оцінками

***Завдання 4:*** Зробіть висновки про використання методів управління потоками в java.

*Спостереження та висновки:*

В даній лабораторній роботі було використано такі методи управління потоками: використання synchronized, wait та notify, ReentrantLock, guarded blocks. Правильне виконання задач цієї лабораторної роботи було б неможливе без використання цих методів контролю потоків і паралельності.

Ключове слово synchronized використовується для забезпечення синхронізації доступу до методів та блоків коду між потоками. Коли потік викликає метод, який має ключове слово synchronized, то він блокує доступ до цього методу для інших потоків, поки поточний потік не завершить виконання методу. Він простий у використанні і забезпечує безпеку від змінення даних декількома потоками одночасно. Але може виникати проблема коли два потоки блокуються, чекаючи один на інший (deadlock) і також потоки мають чекати, доки інший потік завершить роботу з методом.

wait () викликається в потоці і блокує його виконання, поки інший потік не викличе метод notify (). notify () розблокує один потік, який раніше був заблокований методом wait (). Ці методи дозволяють потокам спілкуватися і повідомляти один одного про свій стан, і це знижує час очікування потоків.

Проте для передавання даних між потоками необхідно знаходити інші архітектурні рішення. Наприклад guarded blocks, де ми використовуємо synchronized разом з wait та notify. Його переваги: перешкоджає неправильній поведінці потоків (запобігає deadlock), дозволяє точно визначити умову відновлення, зручний для використання в складних сценаріях.

ReentrantLock надає іншу можливість синхронізації потоків замість synchronized. Він дозволяє потокам блокувати ресурси та використовувати їх в інших потоках. Він немає проблем із заблокуванням потоків та ним можна точніше налаштували поведінку блокування.

**Висновок:**

В ході лабораторної роботи було виконано декілька задач: задача з банком та транзакціями, задача виробник-споживач, електронний журнал. Задачі виконано з використанням паралелізму та контролем потоків.

У звіті наявні скріншоти з прикладами результатів роботи коду, а також опис роботи коду та пояснення результатів.

**Лістинг коду у вигляді GitHub-репозиторію:**

<https://github.com/m4cy43/parallel_programming/tree/master/lab3>