

**Міністерство освіти і науки України**  
**Національний технічний університет 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»

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

**Виконав(ла)**

ІП-14 Сергієнко Ю. В.  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Дифучина О. Ю.  
(прізвище, ім'я, по батькові)

Київ 2024

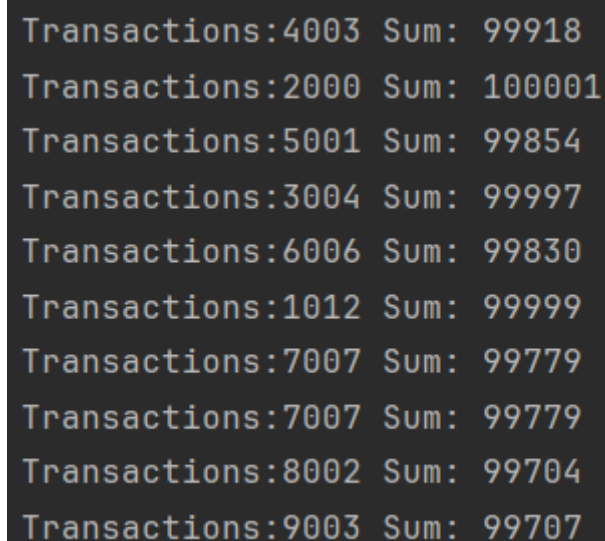
### Комп'ютерний практикум 3

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

#### **Виконання:**

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.**

Для виконання даного завдання необхідно модифікувати код програми для проведення транзакцій. Основною проблемою є неправильний обрахунок трансферу – проблема Race Condition, за якої декілька потоків одночасно використовують один ресурс та неправильно записують результат. За параметрів `NACCOUNTS = 10`, `INITIAL_BALANCE = 1000`, `N_LOG = 1000`, `REPS = 1000` бачимо, що транзакції перемішались (одночасно відбувався інкремент у декількох потоках), а сума відходить від добутку балансу та акаунтів (при додаванні одночасно в один ресурс записується нова сума, що призводить до втрати одного із значень) (рисунок 1).



```
Transactions:4003 Sum: 99918
Transactions:2000 Sum: 100001
Transactions:5001 Sum: 99854
Transactions:3004 Sum: 99997
Transactions:6006 Sum: 99830
Transactions:1012 Sum: 99999
Transactions:7007 Sum: 99779
Transactions:7007 Sum: 99779
Transactions:8002 Sum: 99704
Transactions:9003 Sum: 99707
```

Рисунок 1 – Проблема Race Condition

Дану проблему можна вирішити за допомогою синхронізації роботи потоків, тому спробуємо зробити метод *transfer* синхронізованим. Як бачимо, програма працює правильно (рисунок 2).

```
Transactions:1000 Sum: 100000
Transactions:2000 Sum: 100000
Transactions:3000 Sum: 100000
Transactions:4000 Sum: 100000
Transactions:5000 Sum: 100000
Transactions:6000 Sum: 100000
Transactions:7000 Sum: 100000
Transactions:8000 Sum: 100000
Transactions:9000 Sum: 100000
Transactions:10000 Sum: 100000
```

Рисунок 2 – Сихронізований метод

Спробуємо збільшити кількість операцій до 1000000 (рисунок 3).

```
Transactions:9991000 Sum: 100000
Transactions:9992000 Sum: 100000
Transactions:9993000 Sum: 100000
Transactions:9994000 Sum: 100000
Transactions:9995000 Sum: 100000
Transactions:9996000 Sum: 100000
Transactions:9997000 Sum: 100000
Transactions:9998000 Sum: 100000
Transactions:9999000 Sum: 100000
Transactions:10000000 Sum: 100000
```

Рисунок 3 – Синхронізований метод (1000000 повторів)

Інше рішення проблеми – синхронне виконання потоків за допомогою локерів. У даному випадку використання `ReentrantLock` гарантує виконання лише однієї дії трансферу одночасно, тому не перезапису даних не відбудеться (рисунок 4).

```
Transactions:9992000 Sum: 100000
Transactions:9993000 Sum: 100000
Transactions:9994000 Sum: 100000
Transactions:9995000 Sum: 100000
Transactions:9996000 Sum: 100000
Transactions:9997000 Sum: 100000
Transactions:9998000 Sum: 100000
Transactions:9999000 Sum: 100000
Transactions:10000000 Sum: 100000
```

Рисунок 4 – Використання ReentrantLock

Спробуємо синхронізувати потоки по локеру Object. Фактично, ми досягнемо того ж ефекту, використовуючи блок synchronized (lock) (рисунок 5).

```
Transactions:9992000 Sum: 100000
Transactions:9993000 Sum: 100000
Transactions:9994000 Sum: 100000
Transactions:9995000 Sum: 100000
Transactions:9996000 Sum: 100000
Transactions:9997000 Sum: 100000
Transactions:9998000 Sum: 100000
Transactions:9999000 Sum: 100000
Transactions:10000000 Sum: 100000
```

Рисунок 5 – Використання Object lock

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.**

Основна задача – змінити тип масиву чисел що зчитується на числовий. Протестуємо це передаючи числа від 0 до Size у Consumer. Як бачимо, результат успішний (рисунок 6).

```
MESSAGE RECEIVED: 94  
MESSAGE RECEIVED: 95  
MESSAGE RECEIVED: 96  
MESSAGE RECEIVED: 97  
MESSAGE RECEIVED: 98  
MESSAGE RECEIVED: 99
```

Рисунок 6 – Передача чисел

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.**

Для виконання даного завдання було створено клас GradeBook, де основним атрибутом є список студентів (об'єктів класу Student). Задачею є проставити поточні бали студентам, де максимальною оцінкою є WEEK \* MARK. Кожному асистенту належить одна група, де він може поставити WEEK / 2 оцінок, іншу половину кожній групі виставляє лектор.

Проблема виникає при додаванні балу в журнал студента: якщо лектор та асистент одночасно додають бали студенту, то інколи спрацьовує Memory Consistency Error – лише один результат записується. Для її вирішення необхідно синхронізувати роботу потоків (рисунок 7).

```
Max: 100  
student 74: 95  
student 277: 95  
Got error 2 times!
```

Рисунок 7 – Race condition

Найлегшим способом є використання синхронізованих методів, а результат роботи програми можна бачити на рисунку 8.

```
Max: 100  
Got error 0 times!
```

Рисунок 8 – Робота синхронізованих потоків

4. Зробіть висновки про використання методів управління потоками в java. **10 балів.**

## Висновок

Під час виконання даного комп'ютерного практикуму я закріпив знання та навички щодо розробки паралельних алгоритмів. На прикладі декількох задач було змодельовано помилку Race Condition та знайдено її рішення у вигляді синхронізації потоків за допомогою синхронізованих методів, блоків та локерів. Також було модифіковано програму типу Producer – Consumer для надіслання чисел замість текстових повідомлень.

Код програми доступний на [Github](#).