Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА № 6

з дисципліни «Основи програмування»

Тема: MultiThreading

Виконали:

Студентки групи ІА-31

Горлач Д., Макасєєва М.,

Соколова П.

Перевірив:

Степанов А.

Київ-2023

Тема: MultiTreading

Мета: Ознайомитись з наданими нам матеріалами, пригадати інформацію надану на лекціях з даної теми, також навчившись з минулих лабораторних робіт правильно використовувати знання та реалізовувати за допомогою них завдання, виконати поставлену нам задачу. А саме, скористатись новими знаннями та реалізувати за допомогою них подане завдання.

Хід роботи:

Для того, щоб отримувати час роботи трьох варіантів (формула, "в лоб" з 1 потоком та "в лоб" з декількома потоками), використовуватимемо змінну start, в якій спочатку зберігатимемо поточний час до виконання варіанту в мілісекундах, а потім віднімаємо час після виконання варіанту, і отримуємо в мілісекундах час роботи. Пишемо три методи (), перший з яких використовує формулу для обчислення завдання. Другий метод використовує цикл for в одному потоці. Третій метод приймає параметр кількості потоків, кожний з яких в циклі буде оброблювати свою частину і в блоці synchronized додавати свій результат до загальної суми. Перші 2 методи статичні, а третій не статичний, для того, щоб використовувати блок synchronized, нам потрібен об’єкт, тому цей метод належить до об'єкта Main, на якому і синхронізуємося. Результати виконання усіх трьох варіантів збігаються, а ось час виконання усіх трьох різний.

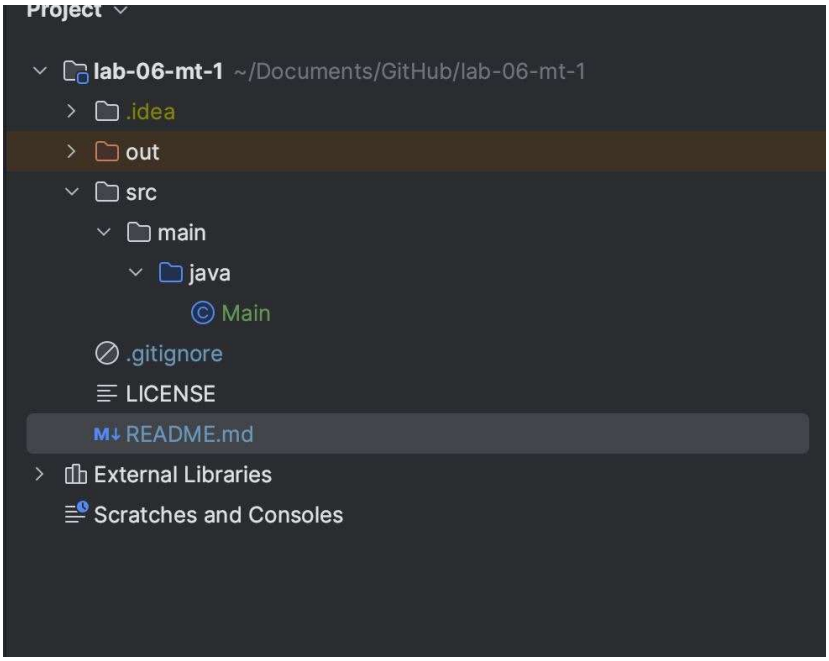
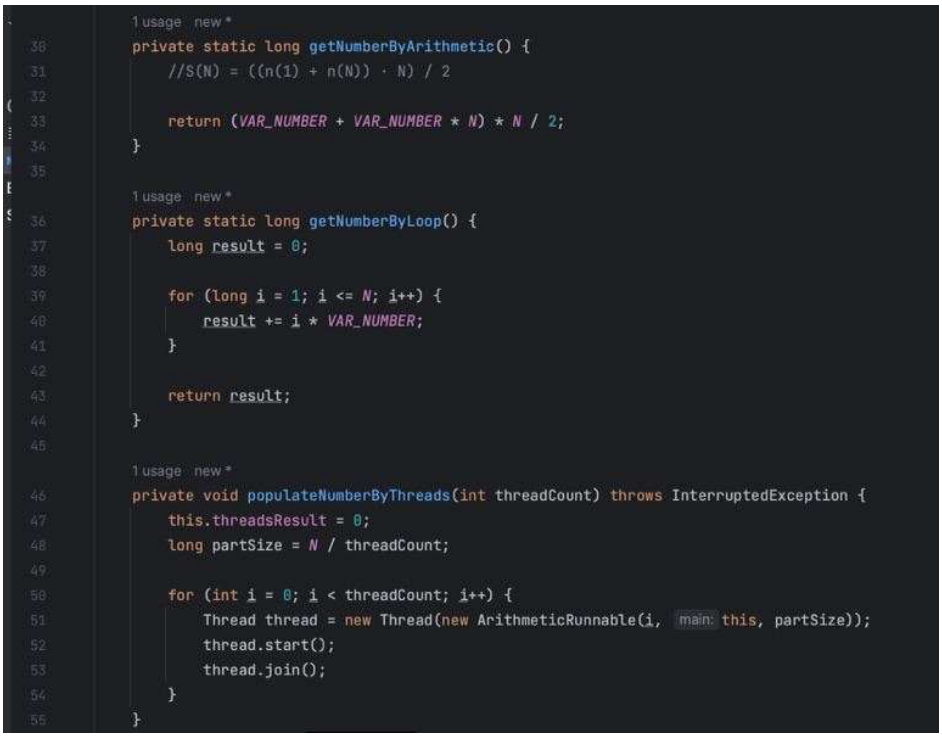


Рис 1. Структура програми

```java
public class Main {

    4 usages
    private static final int VAR_NUMBER = 6;
    4 usages
    private static final long N = 100_000_000;

    3 usages
    private long threadsResult;

    new *
    public static void main(String[] args) throws InterruptedException {
        long start = System.currentTimeMillis();
        System.out.println(getNumberByArithmetic());
        System.out.println(System.currentTimeMillis() - start);

        start = System.currentTimeMillis();
        System.out.println(getNumberByLoop());
        System.out.println(System.currentTimeMillis() - start);

        Main main = new Main();
        List<Integer> threadsCount = List.of(2, 4, 8, 16, 32);

        for (int threadCount : threadsCount) {
            start = System.currentTimeMillis();
            main.populateNumberByThreads(threadCount);
            System.out.println(main.threadsResult);
            System.out.println(System.currentTimeMillis() - start);
        }
    }
```

Рис 2. Main



```
1 usage  new *
private static long getNumberByArithmetic() {
    //S(N) = ((n(1) + n(N)) · N) / 2

    return (VAR_NUMBER + VAR_NUMBER * N) * N / 2;
}

1 usage  new *
private static long getNumberByLoop() {
    long result = 0;

    for (long i = 1; i <= N; i++) {
        result += i * VAR_NUMBER;
    }

    return result;
}

1 usage  new *
private void populateNumberByThreads(int threadCount) throws InterruptedException {
    this.threadsResult = 0;
    long partSize = N / threadCount;

    for (int i = 0; i < threadCount; i++) {
        Thread thread = new Thread(new ArithmeticRunnable(i,  main: this, partSize));
        thread.start();
        thread.join();
    }
}
```

Рис. 3 3 методи

```
/Users/Phoenix/Library/Java/JavaVirtualMachines/openjdk-20.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49355:/Applications/In
300000003000000000
1
300000003000000000
66
300000003000000000
75
300000003000000000
36
300000003000000000
36
300000003000000000
38
300000003000000000
40

Process finished with exit code 0
```

Рис. 4 output 1

```
/Users/Phoenix/Library/Java/JavaVirtualMachines/openjdk-20.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49365:/Applications/In
300000003000000000
0
300000003000000000
64
300000003000000000
73
300000003000000000
37
300000003000000000
36
300000003000000000
38
300000003000000000
42
```

Рис. 5 output 2

Час виконання методів може відрізнятись

Висновок:

Одразу видно, що робота арифметичного методу в нашому випадку найшвидша, тому що ВМ/комп’ютер виконують лише кілька арифметичних операцій. Коли ми виконуємо алгоритм завдяки циклу в одному потоці, час звичайно ж значно зростає, тому що виконуються набагато більше повторних ітерацій. Дуже цікаво було помітити, що коли ми використовуємо 2 треди, час виконання зростає, тому що синхронізація займає набагато більше часу, ніж арифметичні операції. 4 та 8 потоків працюють швидше, тому що час виконання оптимальний між створенням потоків, обсягом роботи і їх синхронізацією. А потім кількість створення потоків займає більше часу ніж виконання самого обчислення. Тож, при розробці програми, необхідно вибирати оптимальну кількість потоків, щоб час їх створення не займав більше часу ніж виконання самого завдання в 1-му чи при меншій кількості потоків.