**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №7

По дисциплине «Информационные технологии и программирование»

Выполнил: Студент группы

БВТ2203

Попов Александр

Москва

2023

**Цель работы:** Знакомство с многопоточностью в ЯП Java.

**Задание 1.** Реализация многопоточной программы для вычисления суммы элементов массива. *Вариант 2.* Создать пул потоков с помощью класса ExecutorService и разделить массив на равные части, каждую из которых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут складываться в главном потоке.

**Задание 2.** Реализация многопоточной программы для поиска наибольшего элемента в матрице. *Вариант 2.* Создать пул потоков с помощью класса ExecutorService и разделить матрицу на равные части, каждую из которых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут сравниваться в главном потоке для нахождения наибольшего элемента.

**Задание 3:** У вас есть склад с товарами, которые нужно перенести на другой склад. У каждого товара есть свой вес. На складе работают 3 грузчика. Грузчики могут переносить товары одновременно, но суммарный вес товаров, которые они переносят, не может превышать 150 кг. Как только грузчики соберут 150 кг товаров, они отправятся на другой склад и начнут разгружать товары. Напишите программу на Java, используя многопоточность, которая реализует данную ситуацию. *Вариант 8.* Использование Semaphore: Используйте семафоры для ограничения доступа к складу и контроля над весом товаров.

**Ход работы**

**Задание 1:**

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

*public* *class* ArraySum {

*public* *static* void main(String[] *args*) {

        int[] data = *new* int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};

        int numThreads = 2;

        Future<Integer>[] results =  *new* Future[numThreads];

        ExecutorService executor = Executors.newFixedThreadPool(numThreads);

        int chunkSize = data.length / numThreads;

        int startIndex = 0;

        int endIndex = chunkSize;

*for* (int i = 0; i < numThreads; i++) {

            int start = startIndex;

            int end = endIndex;

            results[i] = executor.submit(() -> {

                int sum = 0;

*for* (int j = start; j < end; j++) {

                    sum += data[j];

                }

*return* sum;

            });

            startIndex = endIndex;

            endIndex = (i == numThreads - 2) *?* data.length *:* endIndex + chunkSize;

        }

        executor.shutdown();

        int totalSum = 0;

*for* (Future<Integer> result *:* results) {

*try* {

                totalSum += result.get();

            } *catch* (Exception *e*) {

                System.out.println(e);

*return*;

            }

        }

        System.out.println(totalSum);

    }

}

Файл 1. ArraySum.java

В 1 задании я использую ExecutorService для создания пула потоков. 2 потока считают сумму элементов массива, а 3 поток складывает результаты, и результат выводится на экран.

Для потоков, считающих сумму элементов, передаю коллбэк, который проходится по массиву чисел.

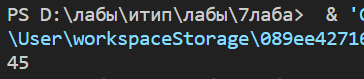


Рис. 1. Результат выполнения программы.

**Задание 2:**

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

*public* *class* MaxElement {

*public* *static* void main(String[] *args*) {

        int[][] data = *new* int[][]{

            {1, 2, 3, 4, 5},

            {5, 6, 65, 2, 1},

            {835, 273, 5, 34}

        };

        int numThreads = data.length;

        Future<Integer>[] results =  *new* Future[numThreads];

        ExecutorService executor = Executors.newFixedThreadPool(numThreads);

*for* (int i = 0; i < numThreads; i++) {

            int index = i;

            results[index] = executor.submit(() -> {

                int max = data[index][0];

*for* (int j = 1; j < data[index].length; j++) {

                    max = (data[index][j] > max) *?* data[index][j] *:* max;

                }

*return* max;

            });

        }

        executor.shutdown();

        int maxElement = data[0][0];

*for* (Future<Integer> result *:* results) {

*try* {

                int number = result.get();

                maxElement = number > maxElement *?* number *:* maxElement;

            } *catch* (Exception *e*) {

                e.printStackTrace();

            }

        }

        System.out.println(maxElement);

    }

}

Файл 2. MaxElement.java

Во втором задании я также создаю пул потоков, которые равномерно разделяют матрицу на массивы, находят в них максимальные элементы и затем передают результат в новый массив. После я прохожу по этому массиву, нахожу максимальный элемент, и вывожу его на экран.

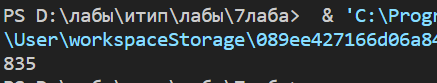


Рис. 2. Результат выполнения программы.

Задание 3:

import java.util.concurrent.Semaphore;

*class* Warehouse {

*private* *final* Semaphore semaphore;

*private* int[] products;

*private* int totalWeight;

*private* Loader[] loaders;

*public* Warehouse(int *totalWeight*, int[] *products*, Loader[] *loaders*) {

        this.totalWeight = *totalWeight*;

        this.semaphore = *new* Semaphore(*loaders*.length);

        this.products = *products*;

        this.loaders = *loaders*;

    }

*public* boolean isEmpty() {

*return* this.products.length == 0;

    }

*private* int popProduct() {

*if* (this.isEmpty()) *return* -1;

        int res = this.products[this.products.length - 1];

        int[] newArr = *new* int[this.products.length-1];

*for* (int i=0; i<products.length - 1; i++) {

            newArr[i] = products[i];

        }

        this.products = newArr;

*return* res;

    }

*public* void start() throws InterruptedException {

*while*(!isEmpty()) {

            int maxWeight = this.totalWeight;

            semaphore.release(3);

*while* (this.totalWeight > 0) {

*for* (Loader loader *:* loaders) {

                    Thread.sleep(1000);

*if* (isEmpty() || this.totalWeight - this.products[this.products.length - 1] < 0) {

                        this.totalWeight = 0;

*break*;

                    }

                    int product = this.popProduct();

                    loader.getProduct(product);

                    this.totalWeight -= product;

                }

            }

            Thread.sleep(2000);

            semaphore.acquire(3);

*for* (Loader loader *:* loaders) {

*new* Thread(loader).start();

            }

            this.totalWeight = maxWeight;

        }

        System.out.println("Warehouse is empty");

    }

}

Файл 3. Warehouse.java.

Для 3 задания мне понадобился класс Warehouse, который содержит в себе массив всех товаров и управляет всеми потоками. Также он имеет метод для удаления последнего элемента из массива товаров и метод start для начала работы.

class Loader implements Runnable {

*private* int productWeight;

*public* Loader() {

        this.productWeight = 0;

    }

*public* void getProduct(int *product*) {

        System.out.println("Loader take a product weight: " + *product*);

        this.productWeight += *product*;

    }

    @Override

*public* void run() {

*try* {

            System.out.println("A loader carries products weighing " + this.productWeight + " kg");

            Thread.sleep(0);

            this.productWeight = 0;

        } *catch* (InterruptedException *e*) {

            Thread.currentThread().interrupt();

        }

    }

}

Файл 4. Loader.java

Класс Loader реализовывает класс Runnable, т.к. в моей задаче Loader будет являться потоком. Метод Run содержит в себе логику заполнения работника товаром и отправкой его на другой склад.

public *class* Main {

*public* *static* void main(String[] *args*) {

        Loader loader1 = *new* Loader();

        Loader loader2 = *new* Loader();

        Loader loader3 = *new* Loader();

        int[] products = *new* int[]{30, 30, 30, 50, 70, 76, 78, 30, 30};

        Loader[] loaders = *new* Loader[]{loader1, loader2, loader3};

        Warehouse warehouse = *new* Warehouse(150, products, loaders);

*try*{

            warehouse.start();

        } *catch* (InterruptedException *e*) {

            System.out.println(e);

        }

    }

}

Файл 5. Main.java.

В классе Main.java выполняется работа программы. Я создаю склад, массив товаров и 3 работников. Затем запускаю процесс переноса товаров.

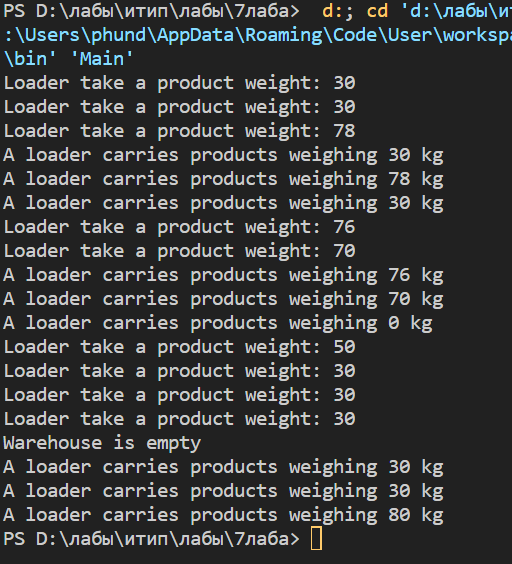


Рис. 3. Результат выполнения программы.

**Выводы**

1. Я научился работать с ExecutorService в ЯП Java. Этот класс позволяет удобно использовать потоки и управлять ими.

2. Я научился работать с Semaphore в ЯП Java. Семафор позволяет ограничивать работу потоков и распределять ресурсы как нам угодно.