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»

Отчет по лабораторной работе №7

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы БПИ2401

Юлдашев Всеволод

Проверил: Харрасов Камиль Раисович

Москва

2025

Цель работы: Изучение основ многопоточного программирования в языке Java. Получение практических навыков работы с классом Thread, интерфейсом Runnable и фреймворком ExecutorService. Освоение механизмов синхронизации потоков, таких как блоки synchronized, методы wait/notify, а также использование высокоуровневых примитивов синхронизации (CyclicBarrier, BlockingQueue) из пакета java.util.concurrent для решения задач взаимодействия потоков.

Ход работы:

=== ЗАДАНИЕ 1: Вычисление суммы элементов массива ===

--- Вариант 1 (Два потока) ---

Файл: Task1Variant1.java

```
public class Task1Variant1 {
    private static int[] array;
    private static int sum1 = 0;
    private static int sum2 = 0;

    public static void main(String[] args) {
        array = new int[100];
        for (int i = 0; i < array.length; i++) {
            array[i] = i + 1;
        }

        int mid = array.length / 2;

        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < mid; i++) {
                sum1 += array[i];
            }
            System.out.println("Поток 1: сумма первой половины = " + sum1);
        });

        Thread thread2 = new Thread(() -> {
```

```
        for (int i = mid; i < array.length; i++) {
            sum2 += array[i];
        }
        System.out.println("Поток 2: сумма второй половины = " + sum2);
    });

    thread1.start();
    thread2.start();

    try {
        thread1.join();
        thread2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    int totalSum = sum1 + sum2;
    System.out.println("Общая сумма элементов массива: " + totalSum);
  }
}
```

--- Вариант 2 (ExecutorService) ---

Файл: Task1Variant2.java

```
import java.util.concurrent.*;
import java.util.ArrayList;
import java.util.List;

public class Task1Variant2 {
  public static void main(String[] args) throws InterruptedException, ExecutionException {
    int[] array = new int[100];
    for (int i = 0; i < array.length; i++) {
        array[i] = i + 1;
```

```
    }

    int numThreads = 4;
    ExecutorService executor = Executors.newFixedThreadPool(numThreads);
    List<Future<Integer>> futures = new ArrayList<>();

    int chunkSize = array.length / numThreads;

    for (int i = 0; i < numThreads; i++) {
        final int start = i * chunkSize;
        final int end = (i == numThreads - 1) ? array.length : (i + 1) * chunkSize;

        Callable<Integer> task = () -> {
            int partialSum = 0;
            for (int j = start; j < end; j++) {
                partialSum += array[j];
            }
            System.out.println("Поток " + Thread.currentThread().getName() +
                               ": сумма с " + start + " по " + (end-1) + " = " + partialSum);
            return partialSum;
        };

        futures.add(executor.submit(task));
    }

    int totalSum = 0;
    for (Future<Integer> future : futures) {
        totalSum += future.get();
    }

    executor.shutdown();
    System.out.println("Общая сумма элементов массива: " + totalSum);
}
```

```
}
```

=== ПРИМЕР ВЫВОДА (ЗАДАНИЕ 1) ===

Поток 1: сумма первой половины = 1275

Поток 2: сумма второй половины = 3775

Общая сумма элементов массива: 5050

=== ЗАДАНИЕ 2: Поиск наибольшего элемента в матрице ===

--- Вариант 1 (Поток на строку) ---

Файл: Task2Variant1.java

```
public class Task2Variant1 {
    private static int[][] matrix;
    private static int[] rowMaxValues;

    public static void main(String[] args) {
        int rows = 5;
        int cols = 10;
        matrix = new int[rows][cols];
        rowMaxValues = new int[rows];

        // Заполнение и вывод матрицы опущены для краткости (см. исходный код)

        Thread[] threads = new Thread[rows];
        for (int i = 0; i < rows; i++) {
            final int rowIndex = i;
            threads[i] = new Thread(() -> {
                int max = matrix[rowIndex][0];
                for (int j = 1; j < matrix[rowIndex].length; j++) {
                    if (matrix[rowIndex][j] > max) {
                        max = matrix[rowIndex][j];
```

```
                }
            }
            rowMaxValues[rowIndex] = max;
            System.out.println("Поток для строки " + rowIndex + ": максимум = " + max);
        });
        threads[i].start();
    }

    // Ожидание и поиск глобального максимума...
  }
}
```

--- Вариант 2 (Пул потоков) ---
Файл: Task2Variant2.java

```
import java.util.concurrent.*;
import java.util.ArrayList;
import java.util.List;

public class Task2Variant2 {
  public static void main(String[] args) throws InterruptedException, ExecutionException {
    // ... Инициализация матрицы ...

    int numThreads = 3;
    ExecutorService executor = Executors.newFixedThreadPool(numThreads);
    List<Future<Integer>> futures = new ArrayList<>();

    int rowsPerThread = (int) Math.ceil((double) rows / numThreads);

    for (int i = 0; i < numThreads; i++) {
      final int startRow = i * rowsPerThread;
      final int endRow = Math.min((i + 1) * rowsPerThread, rows);
```

```
            if (startRow >= rows) break;

            Callable<Integer> task = () -> {
                int max = Integer.MIN_VALUE;
                for (int r = startRow; r < endRow; r++) {
                    for (int c = 0; c < cols; c++) {
                        if (matrix[r][c] > max) max = matrix[r][c];
                    }
                }
                System.out.println("Поток " + Thread.currentThread().getName() +
                                   ": строки " + startRow + "-" + (endRow-1) +
                                   ", максимум = " + max);
                return max;
            };

            futures.add(executor.submit(task));
        }
        // ... Сбор результатов ...
    }
}
```

=== ПРИМЕР ВЫВОДА (ЗАДАНИЕ 2) ===

Матрица:

```
  12   45   99    1 ...
  ...
```

Поток pool-1-thread-1: строки 0-1, максимум = 99

Поток pool-1-thread-2: строки 2-3, максимум = 87

Поток pool-1-thread-3: строки 4-4, максимум = 56

Наибольший элемент в матрице: 99

=== ЗАДАНИЕ 3: Склад (Многопоточная синхронизация) ===

Файл: Task3Warehouse.java

```
// ... (Классы Item и Loader см. в исходном файле) ...

public class Task3Warehouse {
    public static void main(String[] args) {
        BlockingQueue<Item> warehouse = new LinkedBlockingQueue<>();
        // ... Загрузка склада ...

        CyclicBarrier barrier = new CyclicBarrier(3);

        Thread[] loaders = new Thread[3];
        for (int i = 0; i < 3; i++) {
            loaders[i] = new Thread(new Loader(i + 1, warehouse, barrier));
            loaders[i].start();
        }
        // ...
    }
}
```

```
=== ПРИМЕР ВЫВОДА (ЗАДАНИЕ 3) ===
=== СКЛАД ЗАГРУЖЕН ===
Всего товаров: 20
Грузчики начинают работу...

Грузчик 1 взял Ящик №5 (30 кг). Общий вес: 30 кг
Грузчик 2 взял Коробка №12 (40 кг). Общий вес: 70 кг
...
>>> Грузчик 3 готов к отправке! Общий вес: 145 кг
Грузчик 3 ждёт у барьера...
Грузчик 1 ждёт у барьера...
Грузчик 2 ждёт у барьера...
```

=== ВСЕ ГРУЗЧИКИ ОТПРАВИЛИСЬ НА ДРУГОЙ СКЛАД ===
=== РАЗГРУЗКА 145 кг ===

Грузчик 1 завершил работу.
=== ВСЕ ТОВАРЫ ПЕРЕНЕСЕНЫ ===

Вывод: В ходе выполнения лабораторной работы были изучены принципы создания и управления потоками в Java. Реализованы алгоритмы параллельной обработки данных (суммирование элементов массива, поиск максимума в матрице), что позволило распределить вычислительную нагрузку и потенциально ускорить выполнение задач на многоядерных процессорах. Также была успешно решена задача синхронизации доступа к общим ресурсам на примере модели склада. Использование CyclicBarrier обеспечило согласованность действий потоков-грузчиков, а BlockingQueue позволила реализовать потокобезопасную очередь товаров. Работа продемонстрировала преимущества и особенности использования пакета java.util.concurrent.

https://github.com/zevy3/java-labs/tree/main/java7