МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
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

“Многопоточность”

Выполнил: студент группы БПИ2401

ФИО: Орлов Даниил Дмитриевич

Проверил: Харрасов Камиль Раисович

Москва, 2025

Цель работы: изучить принципы многопоточного программирования в языке Java, освоить создание и управление потоками, научиться организовывать параллельное выполнение задач

1 задание, 1 вариант:

```
1   class SumWorker extends Thread {
2       private int[] array;
3       private int startIndex;
4       private int endIndex;
5       private int partialSum;
```

Создаем публичный класс SumWorker, который будет наследовать от Threads.

Далее создаем поле для хранения массива, границы для создания массива, переменную для хранения суммы своей части массива.

```
6       public SumWorker(int[] array, int startIndex, int endIndex) {
7           this.array = array;
8           this.startIndex = startIndex;
9           this.endIndex = endIndex;
10      }
```

Создаем конструктор класса потока, в котором сохраняем ссылку на массив, далее сохраняем границы массива.

```
11      public void run() {
12          partialSum = 0;
13          for (int i = startIndex; i < endIndex; i++) {
14              partialSum += array[i];
15          }
16      }
```

Создаем метод run, внутри которого создадим переменную для суммы перед подсчетом. Далее создаем цикл, который будет перебирать только свою часть массива и добавляет этот элемент к сумме.

```
17      public int getPartialSum() {
18          return partialSum;
19      }
20  }
```

Создаем метод-геттер, который будет возвращать результат работы потока.

```
22  public class SumArrayTwoThreads {
        Run | Debug
23      public static void main(String[] args) throws InterruptedException {
24          int[] numbers = {2, 4, 6, 8, 10, 12};
```

Создаем класс SumArrayTwoThreads, в нем создаем массив с числами.

```
25          SumWorker firstThread = new SumWorker(numbers, startIndex: 0, numbers.length / 2);
26          SumWorker secondThread = new SumWorker(numbers, numbers.length / 2, numbers.length);
```

Создаем первый и второй поток. Первый идет от начала и до середины нашего массива с числами, а второй поток от середины и до конца.

```
27              firstThread.start();
28              secondThread.start();
```

Запускаем первый и второй поток.

```
29              firstThread.join();
30              secondThread.join();
```

Главный поток ждет окончания завершения наших потоков.

```
31          int totalSum = firstThread.getPartialSum() + secondThread.getPartialSum();
32          System.out.println("Сумма элементов массива: " + totalSum);
33      }
34  }
```

Создаем переменную, в которой будем складывать возвращенные суммы и выводим полученный результат.

2 задание 1 вариант:

```
1   class RowMaxFinder extends Thread {
2       private int[] row;
3       private int maxValue;
```

Создаем класс потока, который будет искать максимум в одной строке. Создаем переменную для одной строки матрицы и переменную для максимального элемента строки.

```
7       public void run() {
8           maxValue = row[0];
```

Создаем метод run, который будет служить для прохода по элементам нашей матрицы и переменную для максимального значения, которую изначально приравниваем к первому элементу строки.

```
 9            for (int value : row) {
10                if (value > maxValue) {
11                    maxValue = value;
12                }
13            }
14        }
```

Создаем цикл for-each, который проходит по элементам строки и внутри него сравниваем элемент с максимумом, после чего обновляем максимум.

```
15        public int getMaxValue() {
16            return maxValue;
17        }
18    }
```

Создаем метод-геттер, который возвращает значение максимума.

```
19    public class MaxMatrixByRows {
20        public static void main(String[] args) throws InterruptedException {
21            int[][] matrix = {
22                    {3, 7, 1},
23                    {9, 2, 5},
24                    {4, 6, 8}
25            };
```

Создаем класс MaxMatrixByRows, в котором создаем нашу матрицу.

```
26            RowMaxFinder[] threads = new RowMaxFinder[matrix.length];
```

Создаем по одному массиву потоков на строку.

```
27        for (int i = 0; i < matrix.length; i++) {
28            threads[i] = new RowMaxFinder(matrix[i]);
29            threads[i].start();
30        }
```

Создаем цикл, который будет проходить по строкам нашей матрицы.

```
28              threads[i] = new RowMaxFinder(matrix[i]);
29              threads[i].start();
30          }
```

Каждому потоку передаем свою строку. matrix[i] – строка нашей матрицы, threads[i] – поток для этой строки. Далее запускаем поток. Происходит автоматический вызов run() и начинается поиск максимума строки.

```
31          int globalMax = Integer.MIN_VALUE;
```

Задаем переменную для минимума.

```
32          for (RowMaxFinder thread : threads) {
33              thread.join();
```

Проходимся по элементам массива thread и ожидаем завершения потока.

```
34              if (thread.getMaxValue() > globalMax) {
35                  globalMax = thread.getMaxValue();
36              }
37          }
```

Сравниваем максимум строки и текущий глобальный максимум. Далее обновляем глобальный максимум.

```
38          System.out.println("Максимальный элемент матрицы: " + globalMax);
39      }
40  }
```

Выводим результат

3 задание 7 вариант:

```
1   import java.util.*;
2   import java.util.concurrent.locks.*;
```

Делаем необходимые импорты.

```
3   class Warehouse {
4       private List<Integer> goods;
5       private int currentWeight;
6       private final int MAX_WEIGHT = 150;
7       private ReentrantLock lock;
8       private Condition canLoad;
```

Создаем класс склада. Далее создаем список товаров (их веса), текущий суммарный вес, максимально допустимый вес, блокировку для синхронизации и условие ожидания.

```
9       public Warehouse(List<Integer> goods) {
10          this.goods = new ArrayList<>(goods);
```

Создаем конструктор класса Warehouse и копируем список, чтобы его можно было безопасно изменить.

```
11          this.currentWeight = 0;
12          this.lock = new ReentrantLock();
13          this.canLoad = lock.newCondition();
14      }
```

Задаем текущий суммарный вес равным нулю, создаем блокировки и условия для этой блокировки.

```
15      public void loadGoods(String loaderName) throws InterruptedException {
16          while (true) {
17              lock.lock();
```

Создаем метод загрузки товаров, создаем цикл и захват блокировки.

```
18                  try {
19                      if (goods.isEmpty()) {
20                          return;
21                      }
```

Проверяем остались ли товары. Если нет – поток завершает работу.

```
22                  int weight = goods.get(0);
23                  while (currentWeight + weight > MAX_WEIGHT) {
24                      canLoad.await();
25                  }
```

Берем первый товар. Создаем цикл while с условием “если добавление товара превысит 150 кг”. canLoad.await() – засыпает и освобождает блокировку.

```
26              goods.remove(0);
27              currentWeight += weight;
28              System.out.println(loaderName + " взял товар весом " + weight + ", Текущий вес: " + currentWeight);
```

Удаляем товар со склада и увеличиваем текущий вес. Далее выводим сообщение.

```
29              if (currentWeight == MAX_WEIGHT) {
30                  System.out.println("Отправка на другой склад");
31                  currentWeight = 0;
32                  canLoad.signalAll();
33              }
```

Проверяем достигли ли мы 150 кг и выводим сообщение. Обнуляем текущий вес и пробуждаем всех грузчиков.

```
35              } finally {
36                  lock.unlock();
37              }
38          }
39      }
40  }
```

Освобождаем блокировку в любом случае.

```
41  class Loader extends Thread {
42      private Warehouse warehouse;
43      public Loader(String name, Warehouse warehouse) {
44          super(name);
45          this.warehouse = warehouse;
46      }
```

Создаем класс Loader, который наследуется от потока, значит его обьекты будут потоками. Создаем ссылку на склад, с которым работает грузчик. Далее создаем конструктор, который работает с потоком и складом. Устанавливаем имя потока с помощью обращения к родительскому классу и сохраняем ссылку на склад.

```
47          public void run() {
48              try {
49                  warehouse.loadGoods(getName());
50              } catch (InterruptedException e) {
51                  e.printStackTrace();
52              }
53          }
54      }
```

Создаем метод run. Далее создаем блок try, внутри которого грузчик начинает переносить товары. Ловим исключение прерывание потока.

```
55  public class WarehouseWithLoaders {
56      public static void main(String[] args) {
57          List<Integer> goods = Arrays.asList(40, 30, 50, 20, 60, 40);
58          Warehouse warehouse = new Warehouse(goods);
59          new Loader("Грузчик 1", warehouse).start();
60          new Loader("Грузчик 2", warehouse).start();
61          new Loader("Грузчик 3", warehouse).start();
62      }
63  }
```

Создаем класс WarehouseWithLoaders. Обьявляем и создаем список весов товаров. Далее создаем обьект склада и передаем ему список товаров. Создаем потоки, передаем им имя и склад, после чего запускаем потоки.

Контрольные вопросы:

1. Как реализуется многопоточность в Java?

Многопоточность в Java реализуется с помощью:

- класса Thread;
- интерфейса Runnable;
- интерфейса Callable;
- пула потоков ExecutorService.

Каждый поток выполняется параллельно и имеет собственный путь выполнения.

2. Что такое поток?

Поток - это независимая последовательность выполнения команд внутри одного процесса.
Потоки разделяют память процесса, но выполняются независимо друг от друга.

3. Для чего нужно ключевое слово synchronized?

Ключевое слово synchronized используется для:

- ограничения доступа к общим данным;
- предотвращения одновременного выполнения кода несколькими потоками;
- обеспечения потокобезопасности.

4. Для чего нужно ключевое слово volatile?

Ключевое слово volatile гарантирует, что:

- значение переменной всегда читается из основной памяти;
- изменения переменной сразу видны всем потокам.

Используется для предотвращения проблем видимости данных.

5. Зачем нужно синхронизировать потоки?

Синхронизация потоков необходима для:

- предотвращения гонок данных;
- обеспечения корректности вычислений;
- защиты общих ресурсов от некорректного доступа.

6. Какие есть способы синхронизации потоков?

Основные способы синхронизации:

- synchronized (методы и блоки);
- ReentrantLock;
- volatile;
- wait(), notify(), notifyAll();
- Semaphore;

- CountDownLatch;
- CyclicBarrier.

7. В чем разница между Thread и Runnable?

Thread – является потоком, наследуется, ограничивает наследование.

Runnable – описывает задачу, реализуется, позволяет наследовать другие классы.

8. Какие состояния может иметь поток? Опишите жизненный цикл потока.

Состояния потока:

1. NEW — создан, но не запущен
2. RUNNABLE — выполняется или готов к выполнению
3. BLOCKED — ждёт блокировку
4. WAITING — ждёт неопределённое время
5. TIMED_WAITING — ждёт ограниченное время
6. TERMINATED — завершён

9. Что такое daemon-поток? Как его создать?

Daemon-поток — это фоновый поток, который автоматически завершается при завершении всех пользовательских потоков.

Создание:

thread.setDaemon(true);

10. Как принудительно остановить поток?

Правильный способ — использовать флаг завершения или interrupt().

11. Как работает метод join()? Для чего он используется?

Метод join():

- заставляет один поток ждать завершения другого;
- используется для синхронизации потоков по завершению.

12. Что такое «гонка данных» (race condition)?

Гонка данных — ситуация, когда:

- несколько потоков одновременно изменяют общие данные;
- результат зависит от порядка выполнения потоков.

13. Что такое deadlock? Как его избежать?

Deadlock — взаимная блокировка потоков, когда каждый ждёт ресурс, занятый другим.

Избежать можно:

- соблюдением порядка захвата блокировок;
- использованием таймаутов;
- минимизацией синхронизации.

14. Что такое wait(), notify() и notifyAll()? В каком классе они объявлены?

- wait() — переводит поток в ожидание;
- notify() — пробуждает один поток;
- notifyAll() — пробуждает все потоки.

Объявлены в классе Object.

15. Что такоеThreadPool? КакиереализацииExecutorService есть в Java

ThreadPool — это набор заранее созданных потоков, которые переиспользуются для выполнения задач.

Реализации:

- FixedThreadPool
- CachedThreadPool
- SingleThreadExecutor
- ScheduledThreadPool

Вывод:

В процессе работы были изучены способы создания потоков, организация их взаимодействия, а также методы синхронизации.

Ссылка на репозиторий: https://github.com/qpdle/ITiP.git