

**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»

Дисциплина «Информационные технологии и программирование»

Лабораторная работа №7

«Многопоточность в Java»

Выполнила:

Студентка группы БВТ2303

Морозова Ольга

Цель работы:

Изучение многопоточности и применение полученных знаний на практике на языке программирования Java.

Ход работы:

Вариант 1.

Задание 1.

Напишем программу для вычисления суммы элементов массива.

Для начала создадим класс Sum для хранения переменной суммы и создадим в нём синхронизирующий метод, позволяющий прибавлять к переменной суммы число, а также метод для получения значения суммы.



```
1 package Lab7.Exercise1;
2
3 public class Sum { 4 usages
4     private int sum; 2 usages
5
6     public synchronized void add(int n) { 1 usage
7         sum += n;
8     }
9
10    public int getSum() { 1 usage
11        return sum;
12    }
13 }
```

После создадим класс ThreadSum, наследующий классу Thread, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором и будем проходиться по элементам массива и прибавлять их значения к переменной суммы.

```
WorkTSum.java ThreadSum.java Sum.java
1 package Lab7.Exercise1;
2
3 public class ThreadSum extends Thread { 4 usages
4     private Sum sum; 2 usages
5     private int[] arg; 3 usages
6
7     public ThreadSum(int[] arg, Sum sum) { 2 usages
8         this.sum = sum;
9         this.arg = arg;
10    }
11
12    @Override
13    public void run() {
14        if (arg.length > 0) {
15            for (int i : arg) {
16                sum.add(i);
17            }
18        }
19    }
20 }
```

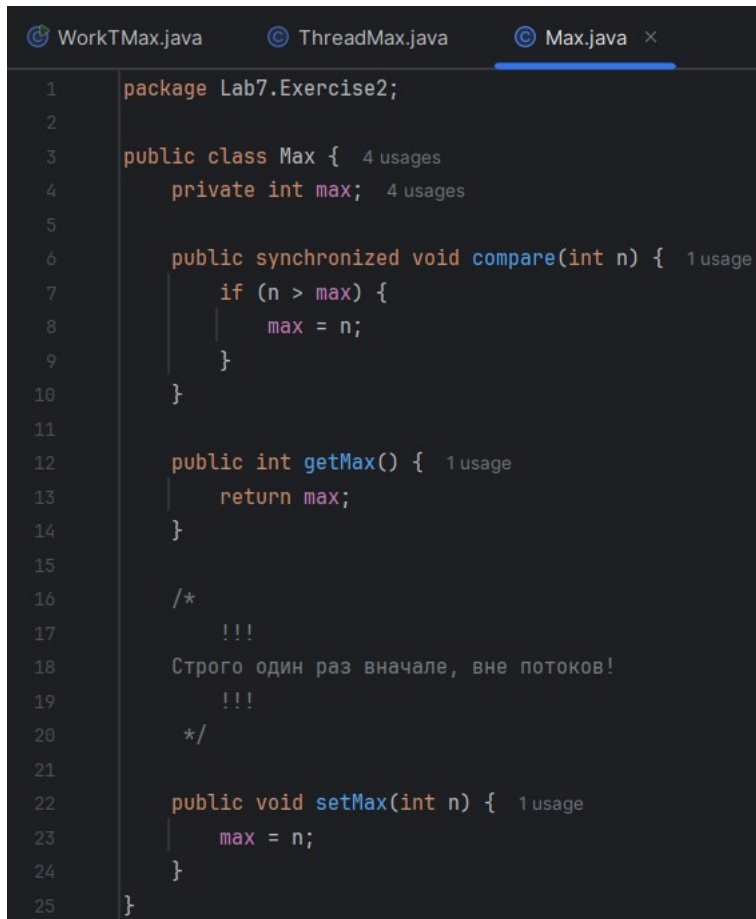
Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода `.start()`, а после завершения из работы сливать обратно методом `.join()`.

```
WorkTSum.java ThreadSum.java Sum.java
1 package Lab7.Exercise1;
2
3 import java.util.Arrays;
4
5 public class WorkTSum {
6     public static void main(String[] args) {
7         int[] arg = {-2, 13, 330, 8, -8, -301, 2};
8         Sum sum = new Sum();
9
10        ThreadSum thread1 = new ThreadSum(Arrays.copyOfRange(arg, 0, arg.length / 2), sum);
11        ThreadSum thread2 = new ThreadSum(Arrays.copyOfRange(arg, arg.length / 2, arg.length), sum);
12        thread1.start();
13        thread2.start();
14        try {
15            thread1.join();
16            thread2.join();
17        } catch (InterruptedException e) {
18            e.printStackTrace();
19        }
20
21        System.out.println(Arrays.toString(arg));
22        System.out.println(sum.getSum());
23    }
24 }
```

Задание 2.

Напишем программу для поиска наибольшего элемента в матрице.

Для начала создадим класс Max для хранения переменной максимума и создадим в нём синхронизирующийся метод, в котором будем сравнивать значение с нынешним максимумом и при необходимости записывать его в переменную, а также геттер и сеттер.



```
1 package Lab7.Exercise2;
2
3 public class Max { 4 usages
4     private int max; 4 usages
5
6     public synchronized void compare(int n) { 1 usage
7         if (n > max) {
8             max = n;
9         }
10    }
11
12    public int getMax() { 1 usage
13        return max;
14    }
15
16    /*
17        !!!
18        Строго один раз вначале, вне потоков!
19        !!!
20    */
21
22    public void setMax(int n) { 1 usage
23        max = n;
24    }
25 }
```

После создадим класс ThreadMax, наследующий классу Thread, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором и будем проходить по элементам строки матрицы для нахождения максимума.

```
WorkTMax.java ThreadMax.java Max.java
1 package Lab7.Exercise2;
2
3 public class ThreadMax extends Thread { 3 usages
4     private Max max; 2 usages
5     private int[] arg; 3 usages
6
7     public ThreadMax(int[] arg, Max max) { 1 usage
8         this.max = max;
9         this.arg = arg;
10    }
11
12    @Override
13    public void run() {
14        if (arg.length > 0) {
15            for (int i : arg) {
16                max.compare(i);
17            }
18        }
19    }
20 }
```

Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода `.start()`, а после завершения из работы сливать обратно методом `.join()`.

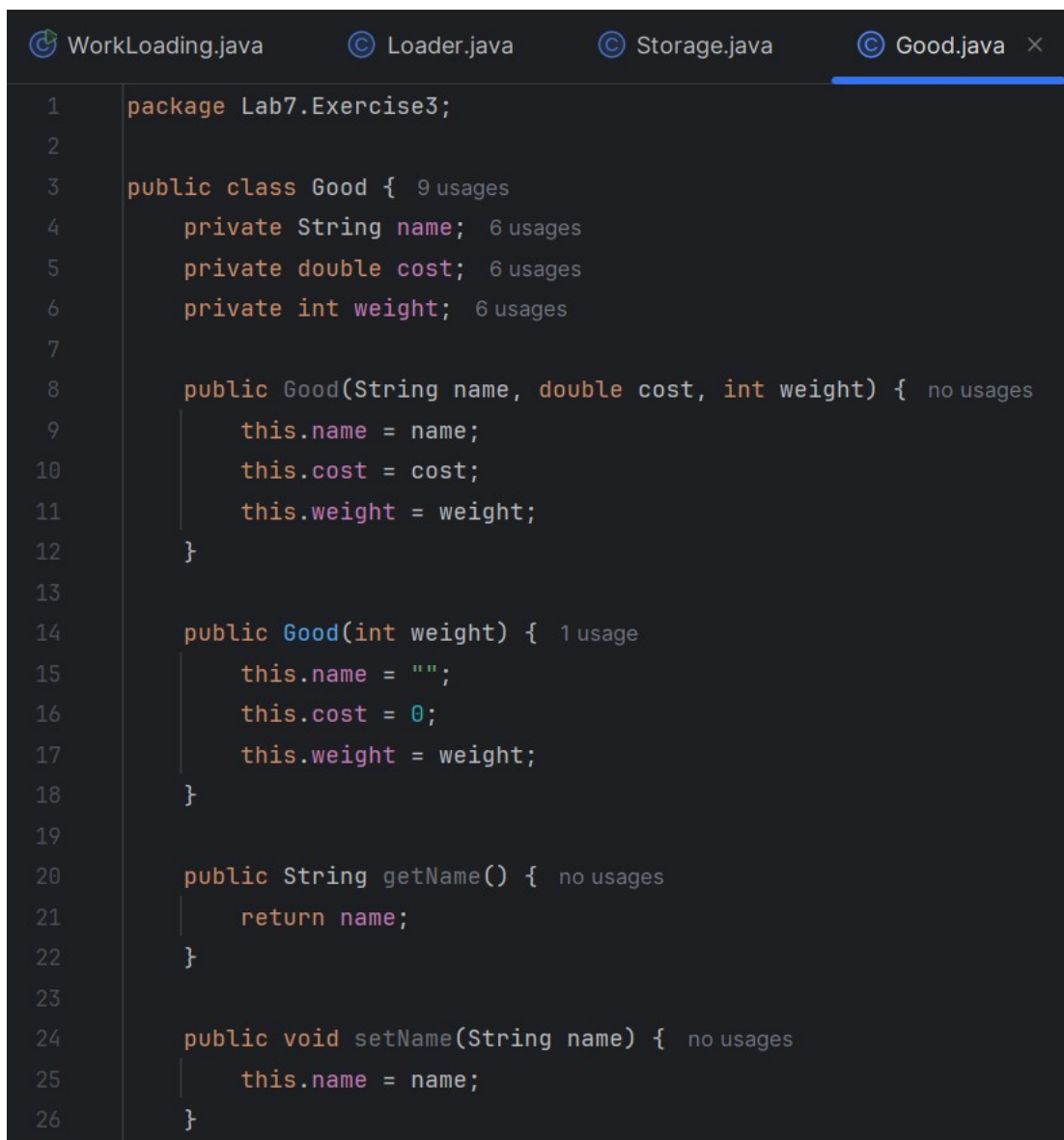
```
WorkTMax.java ThreadMax.java Max.java
1 package Lab7.Exercise2;
2
3 import java.util.Arrays;
4
5 public class WorkTMax {
6     public static void main(String[] args) {
7         int[][] arg = {{-1, -13, 0, -666}, {8, 3, 5, 7}, {18, 6, 9, 3}};
8         Max max = new Max();
9         if (arg.length > 0) {
10             max.setMax(arg[0][0]);
11         }
12
13         ThreadMax[] thread = new ThreadMax[arg.length];
14         for (int i = 0; i < arg.length; i++) {
15             thread[i] = new ThreadMax(arg[i], max);
16             thread[i].start();
17         }
18         try {
19             for (int i = 0; i < arg.length; i++) {
20                 thread[i].join();
21             }
22         } catch (InterruptedException e) {
23             e.printStackTrace();
24         }
25
26         System.out.println(Arrays.deepToString(arg));
27         System.out.println(max.getMax());
28     }
29 }
```

Задание 3.

Напишем программу для реализации следующей задачи:

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.

Для начала создадим класс Good для хранения информации о товаре (наименование, цена, вес) с геттерами и сеттерами.



```
1 package Lab7.Exercise3;
2
3 public class Good { 9 usages
4     private String name; 6 usages
5     private double cost; 6 usages
6     private int weight; 6 usages
7
8     public Good(String name, double cost, int weight) { no usages
9         this.name = name;
10        this.cost = cost;
11        this.weight = weight;
12    }
13
14    public Good(int weight) { 1 usage
15        this.name = "";
16        this.cost = 0;
17        this.weight = weight;
18    }
19
20    public String getName() { no usages
21        return name;
22    }
23
24    public void setName(String name) { no usages
25        this.name = name;
26    }
```

```

28     public double getCost() { no usages
29         return cost;
30     }
31
32     public void setCost(double cost) { no usages
33         this.cost = cost;
34     }
35
36     public int getWeight() { 4 usages
37         return weight;
38     }
39
40     public void setWeight(int weight) { no usages
41         this.weight = weight;
42     }
43
44     public String show() { 1 usage
45         if ((name.equals("")) && (cost == 0)) {
46             return "Бес: " + weight + " кг";
47         } else {
48             return name + ": " + cost + "; " + weight + " кг";
49         }
50     }
51 }

```

Теперь создадим класс Storage для склада, в котором будем хранить ArrayList товаров, а также вес товаров, погруженных в грузовик для перевозки. Напишем методы для того, чтобы взять товар со склада и положить обратно, а также для получения количества товаров на складе и проверку на пустоту.

WorkLoading.java Loader.java **Storage.java** × Good.java

```

1  package Lab7.Exercise3;
2
3  import java.util.ArrayList;
4
5  public class Storage { 8 usages
6      private ArrayList<Good> gds = new ArrayList<>(); 4 usages
7      private int size; 7 usages
8      private Integer tw; 4 usages
9
10     public Storage() { 1 usage
11         size = 0;
12         tw = 0;
13     }
14
15     public Storage(ArrayList<Good> arg) { 1 usage
16         gds.addAll(arg);
17         size = arg.size();
18         tw = 0;
19     }
20
21     public Integer getTw() { 4 usages
22         return tw;
23     }
24
25     public void setTw(Integer tw) { 3 usages
26         this.tw = tw;
27     }

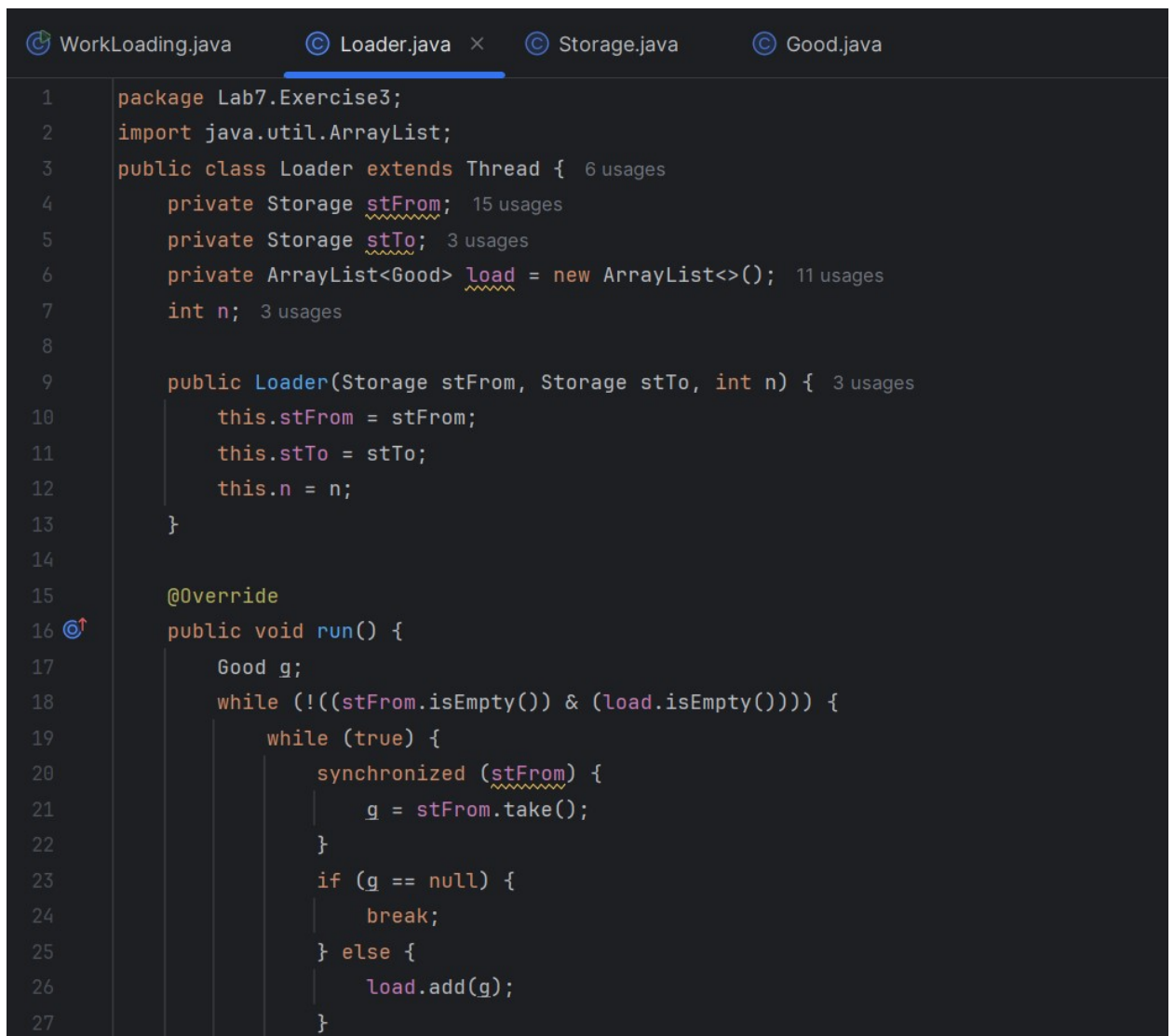
```

```

29     public Good take() { 2 usages
30         if (size > 0) {
31             Good g = gds.getFirst();
32             gds.removeFirst();
33             size--;
34             return g;
35         } else {
36             return null;
37         }
38     }
39
40     public void put(Good g) { 2 usages
41         gds.add(g);
42         size++;
43     }
44
45     public boolean isEmpty() { 2 usages
46         return size <= 0;
47     }
48
49     public int getSize() { 2 usages
50         return size;
51     }
52 }

```


После создадим класс Loader, наследующий классу Thread, для грузчиков, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором сперва грузчик берёт товар с первого склада и, проверяя, может ли тот уместиться по весу в грузовик, либо укладывает его, либо возвращает обратно на склад. Используем синхронизацию с stFrom (первый склад) при заборе товара, а также при сверке переменной tw на возможность добавления товара в грузовик по весу. Затем же происходит выгрузка с синхронизацией с stTo (второй склад). Всё это в цикле while.



```
1 package Lab7.Exercise3;
2 import java.util.ArrayList;
3 public class Loader extends Thread { 6 usages
4     private Storage stFrom; 15 usages
5     private Storage stTo; 3 usages
6     private ArrayList<Good> load = new ArrayList<>(); 11 usages
7     int n; 3 usages
8
9     public Loader(Storage stFrom, Storage stTo, int n) { 3 usages
10         this.stFrom = stFrom;
11         this.stTo = stTo;
12         this.n = n;
13     }
14
15     @Override
16     public void run() {
17         Good g;
18         while (!((stFrom.isEmpty()) & (load.isEmpty()))) {
19             while (true) {
20                 synchronized (stFrom) {
21                     g = stFrom.take();
22                 }
23                 if (g == null) {
24                     break;
25                 } else {
26                     load.add(g);
27                 }
28             }
29         }
30     }
31 }
```

```

28         synchronized (stFrom) {
29             if (stFrom.getTw() + load.getLast().getWeight() > 150) {
30                 stFrom.put(load.getLast());
31                 load.removeLast();
32                 break;
33             } else {
34                 stFrom.setTw(stFrom.getTw() + load.getLast().getWeight());
35                 System.out.println(n + ": " + load.getLast().getWeight());
36             }
37         }
38     }
39     System.out.println(n + ": ОТВОЗИМ. " + stFrom.getTw());
40     synchronized (stTo) {
41         while (!load.isEmpty()) {
42             stTo.put(load.getFirst());
43             synchronized (stFrom) {
44                 stFrom.setTw(stFrom.getTw() - load.getFirst().getWeight());
45             }
46             load.removeFirst();
47         }
48     }
49     synchronized (stFrom) { stFrom.setTw(0); }
50 }
51 }
52 }

```

Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода `.start()`, а после завершения из работы сливать обратно методом `.join()`.

```

WorkLoading.java × Loader.java Storage.java Good.java
1 package Lab7.Exercise3;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5
6 public class WorkLoading {
7     public static void main(String[] args) {
8         int[] arg = new int[] {35, 80, 150, 5, 69, 13, 100, 24, 77};
9         ArrayList<Good> gds = new ArrayList<>();
10        for (int i = 0; i < arg.length; i++) {
11            gds.add(new Good(arg[i]));
12        }
13        Storage stFrom = new Storage(gds);
14        Storage stTo = new Storage();
15
16        Loader loader1 = new Loader(stFrom, stTo, n: 1);
17        Loader loader2 = new Loader(stFrom, stTo, n: 2);
18        Loader loader3 = new Loader(stFrom, stTo, n: 3);
19
20        loader1.start();
21        loader2.start();
22        loader3.start();
23
24        try {
25            loader1.join();
26            loader2.join();
27            loader3.join();
28        } catch (InterruptedException e) {
29            e.printStackTrace();
30        }
31
32        System.out.println(Arrays.toString(arg));
33        System.out.println("На первом складе осталось товаров: " + stFrom.getSize());
34        System.out.println("На втором складе товаров: " + stTo.getSize());
35        System.out.println("Товары на втором складе:");
36        while (!stTo.isEmpty()) {
37            System.out.println(stTo.take().show());
38        }
39    }
40 }

```

Вывод:

Мы изучили многопоточность, существующие возможности достижения многопоточности в Java и применили полученные знания на практике.

GitHub - https://github.com/MiniLynx13/ITaP_Lab7