



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.01 Информатика и вычислительная техника

МАГИСТЕРСКАЯ ПРОГРАММА 09.04.01/07 Интеллектуальные системы анализа,  
обработки и интерпретации больших  
данных

## ОТЧЕТ

по лабораторной работе № 8

Название: Потоки (Threads).

Дисциплина: Языки программирования для работы с большими  
данными

Студент ИУ6-22М

Д. Р. Григорян

Преподаватель

П.В. Степанов  
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

## Цель работы:

Ознакомиться с языком программирования Java, научиться работать с потоками и реализовывать синхронные потоки.

## Выполнение:

### Задача 1.1:

Реализовать многопоточное приложение “Робот”. Надо написать робота, который умеет ходить. За движение каждой его ноги отвечает отдельный поток. Шаг выражается в выводе в консоль LEFT или RIGHT.

### Листинг 1 программы main:

```
import java.util.concurrent.Semaphore;
import java.util.*;
// Реализовать многопоточное приложение "Робот". Надо написать робота,
// который умеет ходить.
// За движение каждой его ноги отвечает отдельный поток.
// Шаг выражается в выводе в консоль LEFT или RIGHT.
public class Main {

    public static void main(String[] args) {
        var robot = new Robot();
        Semaphore sem = new Semaphore(1); // 1 разрешение
        while (robot.active) {
            new Thread(new LeftFootStep("Left",sem)).start();
            new Thread(new RightFootStep("Right",sem)).start();
            System.out.println("Do you want to stop? y/n");
            var in = new Scanner(System.in);
            var letter = in.next().charAt(0);
            if (letter=='n') {
                robot.active = false;
                break;
            }
        }
    }
}

class Robot{
    boolean active = true;
}
```

### Листинг 2 класса потока RightFootStep:

```
import java.util.concurrent.Semaphore;

public class RightFootStep implements Runnable{
    String move;
    Semaphore sem;
    RightFootStep(String Move,Semaphore sem){
        this.move=Move;
        this.sem=sem;
    }

    public void run(){
        try{
```

```

        sem.acquire();
        System.out.println(move);
        Thread.sleep(1000);
    }
    catch (InterruptedException e) {System.out.println(e.getMessage());}
    sem.release();
}
}

```

### Листинг 3 класса потока LeftFootStep:

```

import java.util.concurrent.Semaphore;

public class LeftFootStep implements Runnable{
    String move;
    Semaphore sem;
    LeftFootStep(String Move, Semaphore sem) {
        this.move=Move;
        this.sem=sem;
    }

    public void run() {
        try{
            sem.acquire();
            System.out.println(move);
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {System.out.println(e.getMessage());}
        sem.release();
    }
}

```

Результат приведен на рисунке 1:

```

C:\Users\sonar\.jdk\openjdk-19.0.2\bin
Left
Do you want to stop? y/n
Right
n
Do you want to stop? y/n
Left
Right
n
Do you want to stop? y/n
Left
Right
n
Do you want to stop? y/n
Left
Right
n
Do you want to stop? y/n
Left
Right
y
Process finished with exit code 0

```

Рисунок 1 – Результат выполнения программы

### Задача 1.2:

Реализовать многопоточное приложение “Банк”. Имеется банковский счет. Сделать синхронным пополнение и снятие денежных средств на счет/со счет случайной суммой. При каждой операции (пополнения или снятие) вывести текущий баланс счета. В том случае, если денежных средств недостаточно – вывести сообщение.

### Листинг 4 программы main:

```

import java.util.concurrent.Semaphore;
// Реализовать многопоточное приложение "Банк". Имеется банковский счет.
// Сделать синхронным пополнение и снятие денежных средств на счет/со
// счет случайной суммой.
// При каждой операции (пополнения или снятие) вывести текущий баланс
// счета.
// В том случае, если денежных средств недостаточно - вывести
// сообщение.
public class Main {

    public static void main(String[] args) {
        var account = new Account();
        Semaphore sem = new Semaphore(1); // 1 разрешение
        new Thread(new AdditionThread(account, sem,
"Пополнение",150)).start();
        new Thread(new WithdrawalThread(account, sem, "Снятие",200)).start();
    }
}

```

```

}
class Account{
    int x=0;
}

```

### Листинг 5 класса потока AdditionThread:

```

import java.util.concurrent.Semaphore;

class AdditionThread implements Runnable{

    Account account;
    int price;
    Semaphore sem;
    String name;
    AdditionThread(Account acc, Semaphore sem, String name,int price){
        this.account=acc;
        this.sem=sem;
        this.name=name;
        this.price=price;
    }

    public void run(){
        try{
            System.out.println(name + " is running. Please wait...");
            sem.acquire();
            account.x += price;
            System.out.println("Current account after Addition: "+account.x);
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {System.out.println(e.getMessage());}
        sem.release();
    }
}

```

### Листинг 6 класса потока WithdrawalThread:

```

import java.util.concurrent.Semaphore;

class WithdrawalThread implements Runnable{

    Account account;
    Semaphore sem;
    String name;
    int price;
    WithdrawalThread(Account acc, Semaphore sem, String name,int price){
        this.account=acc;
        this.sem=sem;
        this.name=name;
        this.price=price;
    }

    public void run(){
        try{
            System.out.println(name + " is running. Please wait...");
            sem.acquire();
            if (account.x-price<0) {
                System.out.println("Insufficient funds! ");
            }
            else {
                account.x -= price;
            }
        }
    }
}

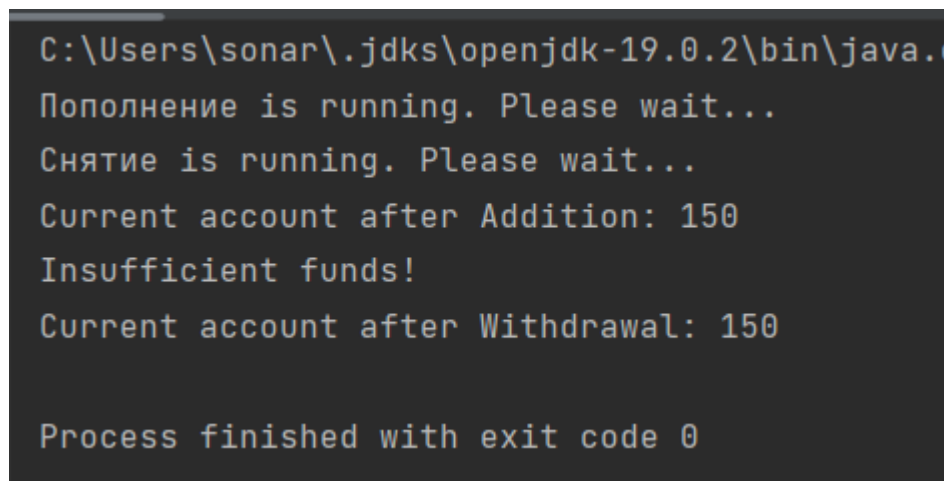
```

```

    }
    System.out.println("Current account after Withdrawal:
"+account.x);
    Thread.sleep(1000);
}
catch (InterruptedException e) {System.out.println(e.getMessage());}
sem.release();
}
}

```

Результат приведен на рисунках 2-3:



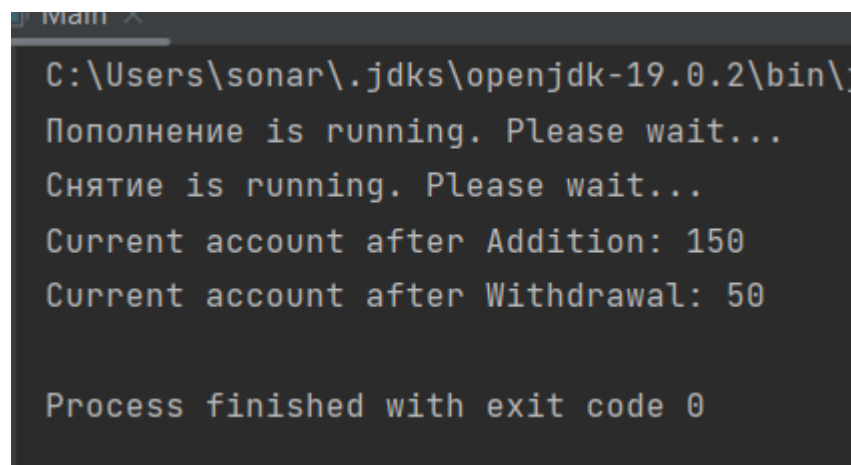
```

C:\Users\sonar\.jdk\openjdk-19.0.2\bin\java.exe
Пополнение is running. Please wait...
Снятие is running. Please wait...
Current account after Addition: 150
Insufficient funds!
Current account after Withdrawal: 150

Process finished with exit code 0

```

Рисунок 2 – Результат списания денег при недостаточных средствах



```

C:\Users\sonar\.jdk\openjdk-19.0.2\bin\java.exe
Пополнение is running. Please wait...
Снятие is running. Please wait...
Current account after Addition: 150
Current account after Withdrawal: 50

Process finished with exit code 0

```

Рисунок 3 – Результат успешного пополнения и списания средств

**Вывод:** в ходе выполнения лабораторной работы были написаны программы согласно выданному заданию, реализованы функции по работе с потоками и их синхронизации.