



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

О Т Ч Е Т

по лабораторной работе № 8

Вариант № 5

Название лабораторной работы: Потоки

Дисциплина: Языки программирования для работы с большими данными

Студент гр. ИУ6-22М

(Подпись, дата)

Р.Г. Гаделия

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

П.В. Степанов

(И.О. Фамилия)

Москва, 2023

Введение

Целью лабораторной работы является приобретение навыков работы с потоками на языке программирования Java.

Практическая часть

Задание 1

Реализовать многопоточное приложение “Магазин”. Вся цепочка: производитель-магазин-покупатель. Пока производитель не поставит на склад продукт, покупатель не может его забрать. Реализовать приход товара от производителя в магазин случайным числом. В том случае, если товара в магазине не хватает– вывести сообщение.

Код написанной программы представлен в листинге 1.

Листинг 1 – Программа для первого задания

```
package com.java.lab;

import java.util.concurrent.Semaphore;

// 3. Реализовать многопоточное приложение “Магазин”. Вся цепочка:
// производитель-магазин-покупатель. Пока производитель не поставит на
// склад продукт, покупатель не может его забрать. Реализовать приход
// товара от производителя в магазин случайным числом. В том случае, если
// товара в магазине не хватает– вывести сообщение.
class Main {
    public static void main(String[] args) {
        var shop = new Shop(10);
        var producerThread = new Thread(() -> {
            while (true) {
                var amount = (int) (Math.random() * 5) + 1;
                shop.put(amount);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        var consumerThread = new Thread(() -> {
            while (true) {
                var amount = (int) (Math.random() * 3) + 1;
                shop.get(amount);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        producerThread.start();
        consumerThread.start();
    }
}

class Shop {
```

```

private int stock;
private final Semaphore producerSemaphore;
private final Semaphore consumerSemaphore;

public Shop(int stock) {
    this.stock = stock;
    producerSemaphore = new Semaphore(1);
    consumerSemaphore = new Semaphore(0);
}

public void put(int amount) {
    try {
        producerSemaphore.acquire();
        stock += amount;
        System.out.println("Producer put new " + amount + " items,
stock: " + stock);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        consumerSemaphore.release();
    }
}

public void get(int amount) {
    try {
        consumerSemaphore.acquire();
        if (stock < amount) {
            System.out.println("Consumer wants " + amount + ", but
there are " + stock + " items in stock");
        } else {
            stock -= amount;
            System.out.println("Consumer bought " + amount + "
items, stock: " + stock);
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        producerSemaphore.release();
    }
}
}

```

Результат выполнения программы показан на рисунке 1.



```

/Users/ivangorshkov/Library/Java/JavaVirtualMachines/openjdk
Producer put new 5 items, stock: 15
Consumer bought 1 items, stock: 14
Producer put new 1 items, stock: 15
Consumer bought 2 items, stock: 13
Producer put new 2 items, stock: 15
Consumer bought 3 items, stock: 12
Producer put new 2 items, stock: 14
Consumer bought 1 items, stock: 13
Producer put new 4 items, stock: 17
Consumer bought 1 items, stock: 16
Producer put new 5 items, stock: 21
Consumer bought 3 items, stock: 18
Producer put new 5 items, stock: 23

```

Рисунок 1 – Результат выполнения программы

Задание 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 снятия) вывести текущий баланс счета. В том случае, если денежных средств недостаточно – вывести сообщение.

Код написанной программы представлен в листинге 2.

Листинг 2 – Программа для второго задания

```
package com.java.lab;

import java.util.Random;
import java.util.concurrent.Semaphore;

public class Main {
    private static final Random random = new Random();

    public static void main(String[] args) {
        Bank bank = new Bank(1000);

        Thread depositThread = new Thread(() -> {
            while (true) {
                try {
                    bank.deposit(random.nextInt(100));
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread withdrawThread = new Thread(() -> {
            while (true) {
                try {
                    bank.withdraw(random.nextInt(100));
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        depositThread.start();
        withdrawThread.start();
    }
}

class Bank {
    private int balance;
    private final Semaphore withdrawSemaphore = new Semaphore(1);
    private final Semaphore depositSemaphore = new Semaphore(1);
}
```

```

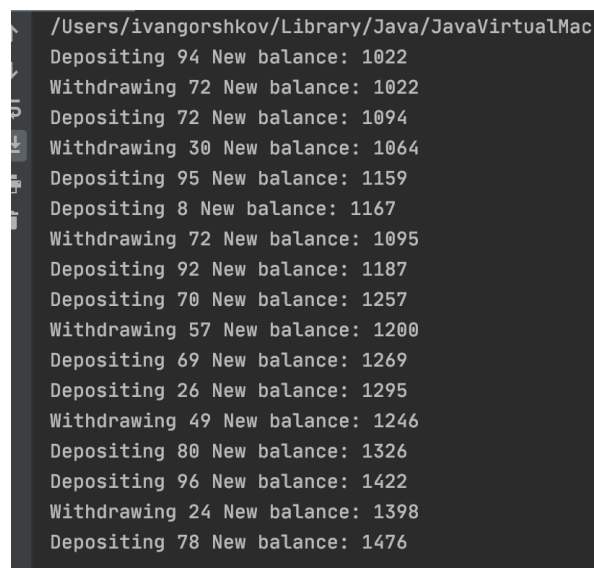
public Bank(int balance) {
    this.balance = balance;
}

public void deposit(int amount) throws InterruptedException {
    depositSemaphore.acquire();
    balance += amount;
    System.out.println("Depositing " + amount + " New balance: " +
balance);
    depositSemaphore.release();
}

public void withdraw(int amount) throws InterruptedException {
    withdrawSemaphore.acquire();
    if (balance < amount) {
        System.out.println("Balance less then amount. Current
balance: " + balance);
        withdrawSemaphore.release();
        return;
    }
    balance -= amount;
    System.out.println("Withdrawing " + amount + " New balance: "
+ balance);
    withdrawSemaphore.release();
}
}

```

Результат выполнения программы показан на рисунке 2.



```

/Users/ivangorshkov/Library/Java/JavaVirtualMac
Depositing 94 New balance: 1022
Withdrawing 72 New balance: 1022
Depositing 72 New balance: 1094
Withdrawing 30 New balance: 1064
Depositing 95 New balance: 1159
Depositing 8 New balance: 1167
Withdrawing 72 New balance: 1095
Depositing 92 New balance: 1187
Depositing 70 New balance: 1257
Withdrawing 57 New balance: 1200
Depositing 69 New balance: 1269
Depositing 26 New balance: 1295
Withdrawing 49 New balance: 1246
Depositing 80 New balance: 1326
Depositing 96 New balance: 1422
Withdrawing 24 New balance: 1398
Depositing 78 New balance: 1476

```

Рисунок 2 – Результат выполнения программы

Вывод: В результате выполнения лабораторной работы были приобретены навыки работы с потоками на языке программирования Java.