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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ 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

Название: Потоки (Threads)

Дисциплина: Языки программирования для работы с большими данными

Студент ИУ6-22М (Группа) _____ (Подпись, дата) А.М. Панфилкин (И.О. Фамилия)

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

Москва, 2022 г.

Весь приведенный ниже код также доступен в следующем репозитории:

https://github.com/SilkSlime/iu6plfbd

Задание 1: 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 1 – Задание 1

```
package l8;

import java.util.Random;

public class Bank {
    // Общая для потоков переменная
    private int balance = 0;

    // Метод, который вызывается потоком для пополнения счета
    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println("Deposited " + amount + ". Balance is now " + balance + ".");
    }

    // Метод, который вызывается потоком для снятия денег со счета
    public synchronized void withdraw(int amount) {
        if (balance < amount) {
            System.out.println("Not enough funds to withdraw " + amount + ". Balance is " +
balance + ".");
        } else {
            balance -= amount;
            System.out.println("Withdrew " + amount + ". Balance is now " + balance + ".");
        }
    }

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

        // Создаем 10 потоков, которые будут пополнять счет
        for (int i = 0; i < 10; i++) {
            new Thread(() -> {
                Random random = new Random();
                int amount = random.nextInt(100);
                bank.deposit(amount);
            }).start();
        }

        // Создаем 10 потоков, которые будут снимать деньги со счета
        for (int i = 0; i < 10; i++) {
            new Thread(() -> {
```

```
                Random random = new Random();
                int amount = random.nextInt(100);
                bank.withdraw(amount);
            }).start();
        }
    }
}
```

Задание 2: 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.

Листинг 2 – Задание 2

```
package l8;

import java.util.Random;

public class Shop {
    private int stock;

    public Shop(int initialStock) {
        stock = initialStock;
    }

    // Вызывается производителем
    public synchronized void receiveGoods(int quantity) {
        stock += quantity;
        System.out.println(Thread.currentThread().getName() + " received " + quantity + "
goods. Stock: " + stock);
        // notifyAll - пробуждает все потоки, которые ждут этого объекта
        notifyAll();
    }

    // Вызывается покупателем
    public synchronized void buyGoods(int quantity) {
        while (stock < quantity) {
            try {
                System.out.println(Thread.currentThread().getName() + " wants to buy " +
quantity + " goods but stock is " + stock + ". Waiting for more goods to arrive.");
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        stock -= quantity;
        System.out.println(Thread.currentThread().getName() + " bought " + quantity + " goods.
Stock: " + stock);
    }
```

```
    public static void main(String[] args) {
        // Создаем магазин с начальным запасом 10 товаров
        Shop shop = new Shop(10);

        // Создаем производителя и покупателей
        Manufacturer manufacturer = new Manufacturer(shop);
        Thread manufacturerThread = new Thread(manufacturer);

        // Запускаем производителя
        manufacturerThread.start();

        // Запускаем покупателей
        Buyer buyer1 = new Buyer(shop, 5);
        Thread buyer1Thread = new Thread(buyer1);
        buyer1Thread.start();

        Buyer buyer2 = new Buyer(shop, 7);
        Thread buyer2Thread = new Thread(buyer2);
        buyer2Thread.start();
    }

    // implements Runnable - означает, что класс реализует интерфейс Runnable
    // Runnable - интерфейс, который содержит метод run()
    // run() - метод, который будет выполняться в отдельном потоке
    private static class Manufacturer implements Runnable {
        private Shop shop;
        private Random random;

        public Manufacturer(Shop shop) {
            this.shop = shop;
            random = new Random();
        }

        @Override
        public void run() {
            while (true) {
                try {
                    int quantity = random.nextInt(10) + 1;
                    Thread.sleep(2000);
                    shop.receiveGoods(quantity);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    private static class Buyer implements Runnable {
        private Shop shop;
        private int quantity;

        public Buyer(Shop shop, int quantity) {
            this.shop = shop;
```

```
            this.quantity = quantity;
        }

        @Override
        public void run() {
            while (true) {
                shop.buyGoods(quantity);
                try {
                    Thread.sleep(1000); // simulate shopping time
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Вывод: В ходе лабораторной работы мы изучили концепцию потоков (Threads) в Java и применили их на практике в двух приложениях: "Банк" и "Магазин". В первом приложении мы создали многопоточное приложение, которое позволяет пополнять и снимать деньги со счета, при этом выводится текущий баланс счета. Мы также сделали операции синхронными для избежания ошибок. Во втором приложении мы реализовали цепочку "производитель-магазин-покупатель", где производитель должен поставить товар на склад, прежде чем покупатель может его купить. Если товара не хватает, выводится сообщение. В результате выполнения лабораторной работы мы узнали, как правильно использовать потоки в Java и как они могут помочь в решении многозадачных задач.