Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №7**

**Дисциплина: Платформо-независимое программирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Агаджанян

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. И. Шиян

**Тема**: потоки Executable

**Задание**: В порт грузовиками доставляются грузы одинаковыми по грузоподъемности грузовиками – 10т, которые сгружаются на 5 платформ. К платформам причаливают грузовые суда разного тоннажа (100, 120, 150т), загружаются до тех пор, пока не наберут груза по предельной возможности, затем освобождают место другому судну для загрузки.

**Описание работы программы:**

1. Создается экземпляр класса Port.
2. Создаются два ExecutorService для грузовиков и кораблей.
3. Запускаются потоки грузовиков и кораблей с использованием ExecutorService.
4. Грузовики постоянно пытаются разгрузить груз на доступные платформы. Если все платформы заняты, грузовик ждет, пока одна из них не освободится.
5. Корабли постоянно пытаются загрузить груз из порта. Если груза недостаточно, корабль ждет, пока его не накопится достаточно.
6. Синхронизация доступа к платформам и общему количеству груза осуществляется с помощью Lock и Condition.
7. Обработка InterruptedException необходима для корректного завершения потоков при прерывании. После перехвата исключения Thread.currentThread().interrupt() восстанавливает флаг прерывания, а return завершает выполнение метода unloadTruck или loadShip.

**Листинг:**  
import java.util.LinkedList;

import java.util.Queue;

import java.util.Random;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class Port {

    private static final int NUM\_PLATFORMS = 5;

    private static final int TRUCK\_CAPACITY = 10;

    private static final int[] SHIP\_CAPACITIES = {100, 120, 150};

    private final Queue<Integer> platforms = new LinkedList<>(); // Queue для представления платформ

    private final Lock lock = new ReentrantLock(); // Lock для синхронизации доступа к платформам и грузу

    private final Condition platformAvailable = lock.newCondition(); // Condition для ожидания доступности платформы

    private int totalCargo = 0; // Общее количество груза в порту

    public Port() {

        // Изначально все платформы свободны

        for (int i = 0; i < NUM\_PLATFORMS; i++) {

            platforms.offer(i);

        }

    }

    public static void main(String[] args) {

        Port port = new Port();

        // Создаем ExecutorService для грузовиков и кораблей

        ExecutorService truckExecutor = Executors.newFixedThreadPool(10); // 10 грузовиков

        ExecutorService shipExecutor = Executors.newFixedThreadPool(3);  // 3 корабля

        // Запускаем грузовики

        for (int i = 0; i < 10; i++) {

            truckExecutor.execute(port::unloadTruck);

        }

        // Запускаем корабли

        for (int i = 0; i < 3; i++) {

            int shipCapacity = SHIP\_CAPACITIES[new Random().nextInt(SHIP\_CAPACITIES.length)];

            shipExecutor.execute(() -> port.loadShip(shipCapacity));

        }

    }

    // Метод для разгрузки грузовика

    public void unloadTruck() {

        Random random = new Random();

        while (true) {

            try {

                Thread.sleep(random.nextInt(1000)); // Имитация времени в пути грузовика

                lock.lock();

                try {

                    while (platforms.isEmpty()) {

                        System.out.println("Грузовик ждет доступную платформу...");

                        platformAvailable.await(); // Ждем, пока платформа не освободится

                    }

                    int platformNumber = platforms.poll(); // Получаем номер свободной платформы

                    System.out.println("Грузовик разгружается на платформе " + platformNumber);

                    totalCargo += TRUCK\_CAPACITY;

                    System.out.println("Добавлено груза: " + TRUCK\_CAPACITY + " тонн.  Всего груза: " + totalCargo + " тонн.");

                    // Уведомляем корабли, что появился новый груз

                    platformAvailable.signalAll();

                    Thread.sleep(random.nextInt(500)); // Имитация времени разгрузки

                    platforms.offer(platformNumber); // Освобождаем платформу

                    platformAvailable.signalAll(); // Уведомляем другие грузовики и корабли о доступности платформы

                    System.out.println("Платформа " + platformNumber + " освобождена.");

                } finally {

                    lock.unlock();

                }

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt(); // Очень важно восстановить флаг прерывания!

                return; // Завершаем поток

            }

        }

    }

    // Метод для загрузки корабля

    public void loadShip(int shipCapacity) {

        Random random = new Random();

        while (true) {

            try {

                Thread.sleep(random.nextInt(2000)); // Имитация времени прибытия корабля

                lock.lock();

                try {

                    System.out.println("Корабль прибыл. Вместимость: " + shipCapacity + " тонн.  Текущий груз в порту: " + totalCargo + " тонн.");

                    while (totalCargo < 10) { // Небольшое изменение: ждем хотя бы 10 тонн груза перед началом загрузки

                         System.out.println("Корабль ждет груз.  Текущий груз: " + totalCargo + " тонн.");

                         platformAvailable.await(); // Ждем, пока не появится достаточно груза

                     }

                    int loadedCargo = Math.min(shipCapacity, totalCargo); // Определяем, сколько груза можно загрузить

                    totalCargo -= loadedCargo;

                    System.out.println("Корабль загрузил " + loadedCargo + " тонн. Осталось груза: " + totalCargo + " тонн.");

                    Thread.sleep(random.nextInt(1000)); // Имитация времени загрузки

                    System.out.println("Корабль отплыл.");

                    platformAvailable.signalAll();  // Уведомляем грузовики о возможном появлении места на платформах

                } finally {

                    lock.unlock();

                }

            } catch (InterruptedException e) {

                Thread.currentThread().interrupt();

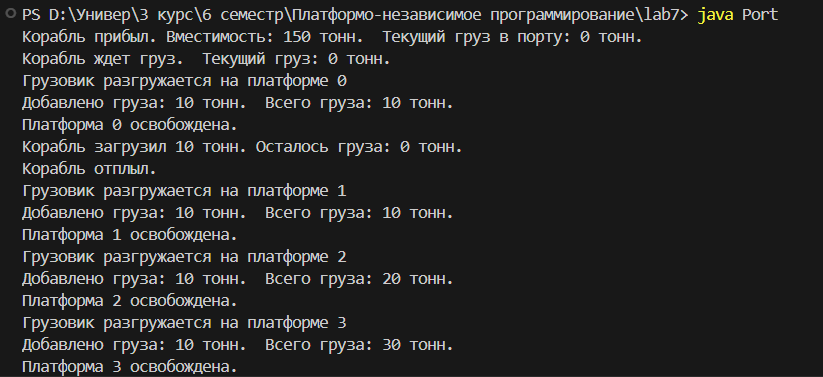
                return; // Завершаем поток

            }

        }

    }

}



Вывод: в ходе выполнения лабораторной работы были получены навыки работы с многопоточностью в Java, использования ExecutorService для управления потоками, а также применения Lock и Condition для синхронизации доступа к общим ресурсам. Была разработана модель порта, демонстрирующая принципы конкурентного доступа к ресурсам и необходимость обеспечения корректной синхронизации. Пример демонстрирует, как можно эффективно использовать потоки для решения задач, требующих параллельной обработки данных или имитации параллельных процессов.