МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе № 7

Выполнил: студент группы БПИ2401
Трухина Анастасия Александровна
Проверил: Харрасов Камиль Раисович

Москва,
2025

Оглавление

Цель работы:
Освоить теоретические основы и получить практические навыки работы с коллекциями (Collections Framework) в языке Java. Изучить иерархию интерфейсов (Collection, List, Set, Map, Queue) и их основные реализации (ArrayList, LinkedList, HashSet, HashMap и др.). Научиться применять дженерики для обеспечения типобезопасности кода. Сравнить характеристики и сферы применения различных коллекций для осознанного выбора оптимальной структуры данных под конкретную задачу. Приобрести умение использовать итераторы для безопасного обхода коллекций и изучить базовые принципы сортировки с помощью интерфейсов Comparable и Comparator.

Задание:.
Задание 1. Реализация многопоточнойпрограммы для вычисления сум мы элементов массива.
Вариант 1. Создать два потока, которые будут вычислять сумму элементов массива по половинкам, после чего результаты будут складываться в главном потоке.

Вариант 2. Создать пул потоков с помощью класса Executor Service и разделить массив на равные части, каждую из которых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут складываться в главном потоке.

Задание 2. Реализация многопоточной программы для поиска наиболь шего элемента в матрице.
Вариант 1. Создать несколько потоков, каждый из которых будет обрабатывать свою строку матрицы. После завершения работы всех потоков результаты будут сравниваться в главном потоке для нахождения наибольшего элемента.

Вариант 2. Создать пул потоков с помощью класса Executor Service и разделить матрицу на равные части, каждую из кото рых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут сравниваться в главном потоке для нахождения наибольшего элемента.

Задание 3. У вас есть склад с товарами, которые нужно перенести на другой склад. У каждого товара есть свой вес. На складе ра ботают 3 грузчика. Грузчики могут переносить товары одно временно, но суммарный вес товаров, переносимый ими за одну итерацию, не может превышать 150 кг. Как только грузчики со берут 150 кг товаров, они отправятся на другой склад и начнут разгружать товары. Напишите программу на Java, используя многопоточность, которая реализует данную ситуацию.

Варианты

1. Использование Thread. Создайте классы Товар, Склад, и Грузчик. Каждый грузчик должен быть представлен в виде отдельного потока.

2. Использование Runnable. Создайте интерфейс Грузчик и реализуйте его в классе LoaderRealization. Используйте Executor Service для управления потоками.

3. Использование Lock и Condition. Используйте блокировки и условия для синхронизации работы грузчиков.

4. Использование Semaphore. Используйте семафоры для ограничения доступа к складу и контроля над весом товаров.

5. Использование CompletableFuture. Используйте Complet ableFuture для асинхронного выполнения задачи переноса това ров.

6. Использование ForkJoinPool. Разделите склад на подзада чи и используйте Fork-Join-пул для обработки этих подзадач.

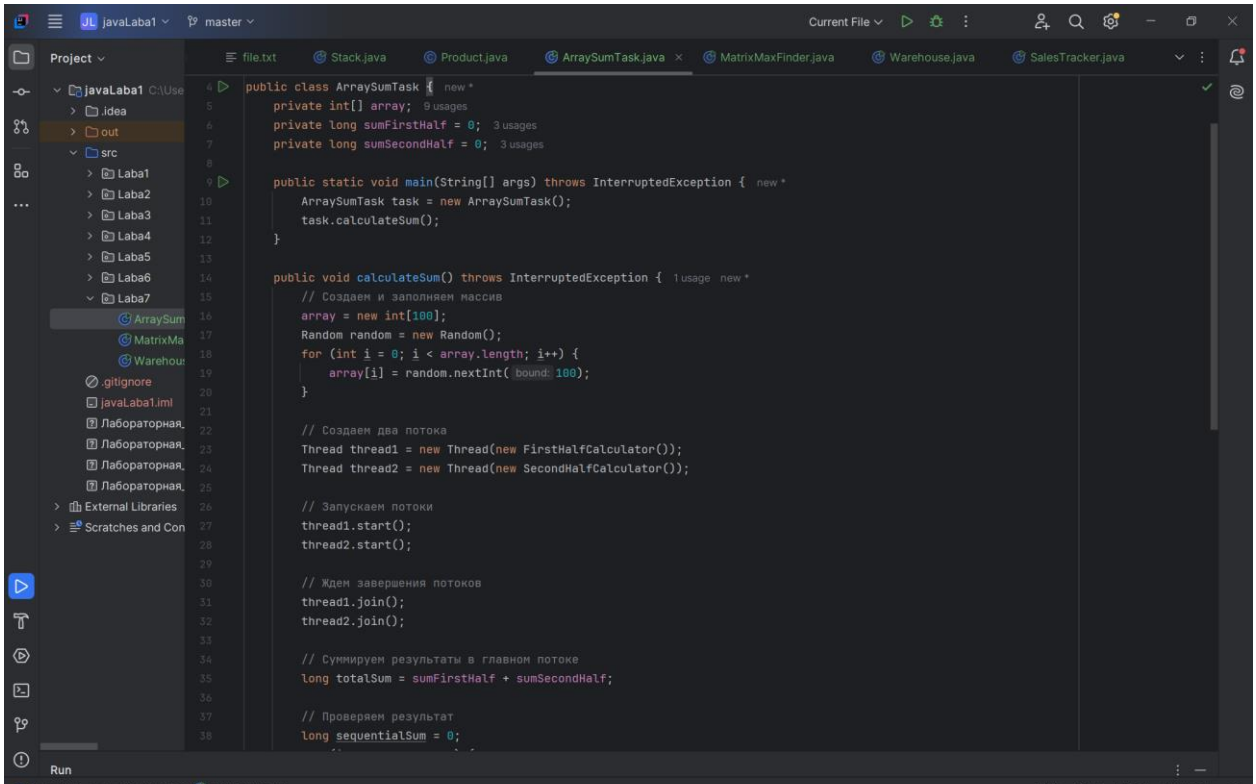
7. Использование ReentrantLock и Condition. Используйте ре ентерабельныеблокировкидля управлениядоступомк ресурсам.

8. ИспользованиеCountDownLatch. ИспользуйтеCountDown Latch для синхронизации начала и завершения переноса товаров.

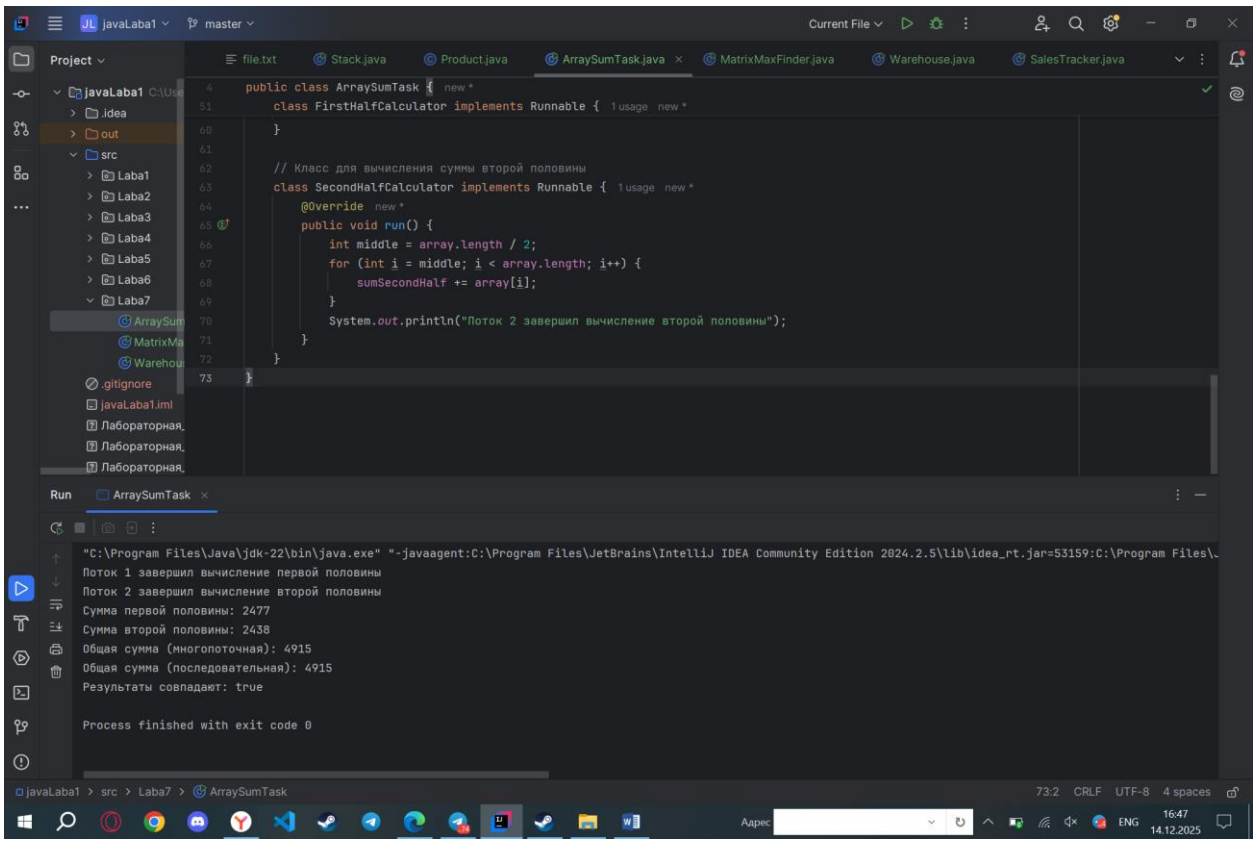
9. Использование CyclicBarrier. Используйте барьеры для синхронизации грузчиков перед отправлением на другой склад. 10. Использование Executor и CompletionService.Используй те Executor для управления потоками и CompletionService для по лучения результатов выполнения.

Основная часть

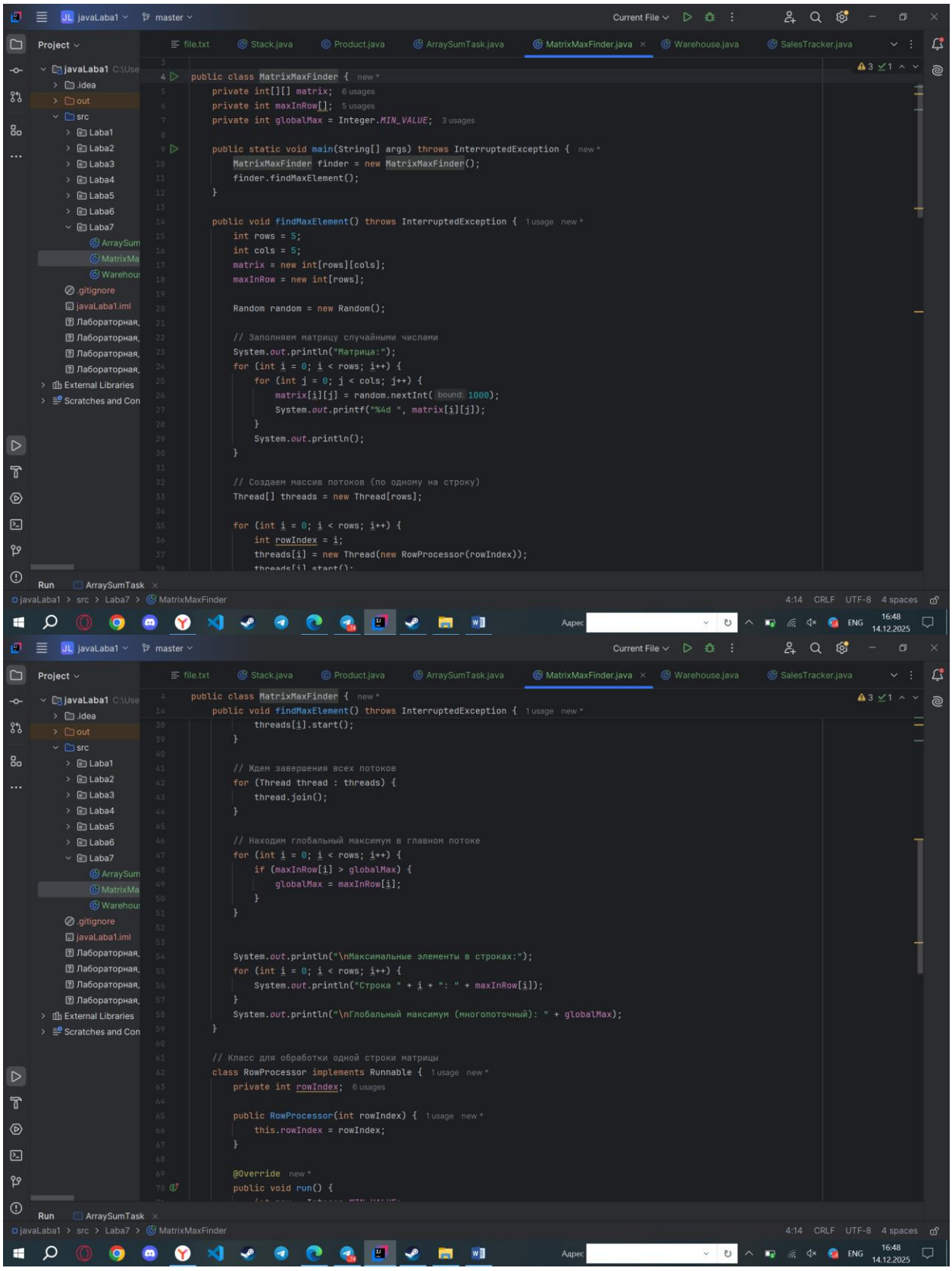
Задание 1:



```java
public class ArraySumTask {
    private int[] array;
    private long sumFirstHalf = 0;
    private long sumSecondHalf = 0;

    public static void main(String[] args) throws InterruptedException {
        ArraySumTask task = new ArraySumTask();
        task.calculateSum();
    }

    public void calculateSum() throws InterruptedException {
        // Создаем и заполняем массив
        array = new int[100];
        Random random = new Random();
        for (int i = 0; i < array.length; i++) {
            array[i] = random.nextInt( bound: 100);
        }

        // Создаем два потока
        Thread thread1 = new Thread(new FirstHalfCalculator());
        Thread thread2 = new Thread(new SecondHalfCalculator());

        // Запускаем потоки
        thread1.start();
        thread2.start();

        // Ждем завершения потоков
        thread1.join();
        thread2.join();

        // Суммируем результаты в главном потоке
        long totalSum = sumFirstHalf + sumSecondHalf;

        // Проверяем результат
        long sequentialSum = 0;
        for (int value : array) {
            sequentialSum += value;
        }

        System.out.println("Сумма первой половины: " + sumFirstHalf);
        System.out.println("Сумма второй половины: " + sumSecondHalf);
        System.out.println("Общая сумма (многопоточная): " + totalSum);
        System.out.println("Общая сумма (последовательная): " + sequentialSum);
        System.out.println("Результаты совпадают: " + (totalSum == sequentialSum));
    }

    // Класс для вычисления суммы первой половины
    class FirstHalfCalculator implements Runnable {
        @Override
        public void run() {
            int middle = array.length / 2;
            for (int i = 0; i < middle; i++) {
                sumFirstHalf += array[i];
            }
            System.out.println("Поток 1 завершил вычисление первой половины");
        }
    }

    // Класс для вычисления суммы второй половины
    class SecondHalfCalculator implements Runnable {
        @Override
        public void run() {
            int middle = array.length / 2;
            for (int i = middle; i < array.length; i++) {
                sumSecondHalf += array[i];
            }
            System.out.println("Поток 2 завершил вычисление второй половины");
        }
    }
```


javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
public class ArraySumTask {
class FirstHalfCalculator implements Runnable {
// Класс для вычисления суммы второй половины
class SecondHalfCalculator implements Runnable {
@Override
public void run() {
int middle = array.length / 2;
for (int i = middle; i < array.length; i++) {
sumSecondHalf += array[i];
System.out.println("Поток 2 завершил вычисление второй половины");
Run
ArraySumTask
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=53159:C:\Program Files\
Поток 1 завершил вычисление первой половины
Поток 2 завершил вычисление второй половины
Сумма первой половины: 2477
Сумма второй половины: 2438
Общая сумма (многопоточная): 4915
Общая сумма (последовательная): 4915
Результаты совпадают: true
Process finished with exit code 0
javaLaba1 > src > Laba7 > ArraySumTask
73:2 CRLF UTF-8 4 spaces
Адрес
ENG
16:47
14.12.2025

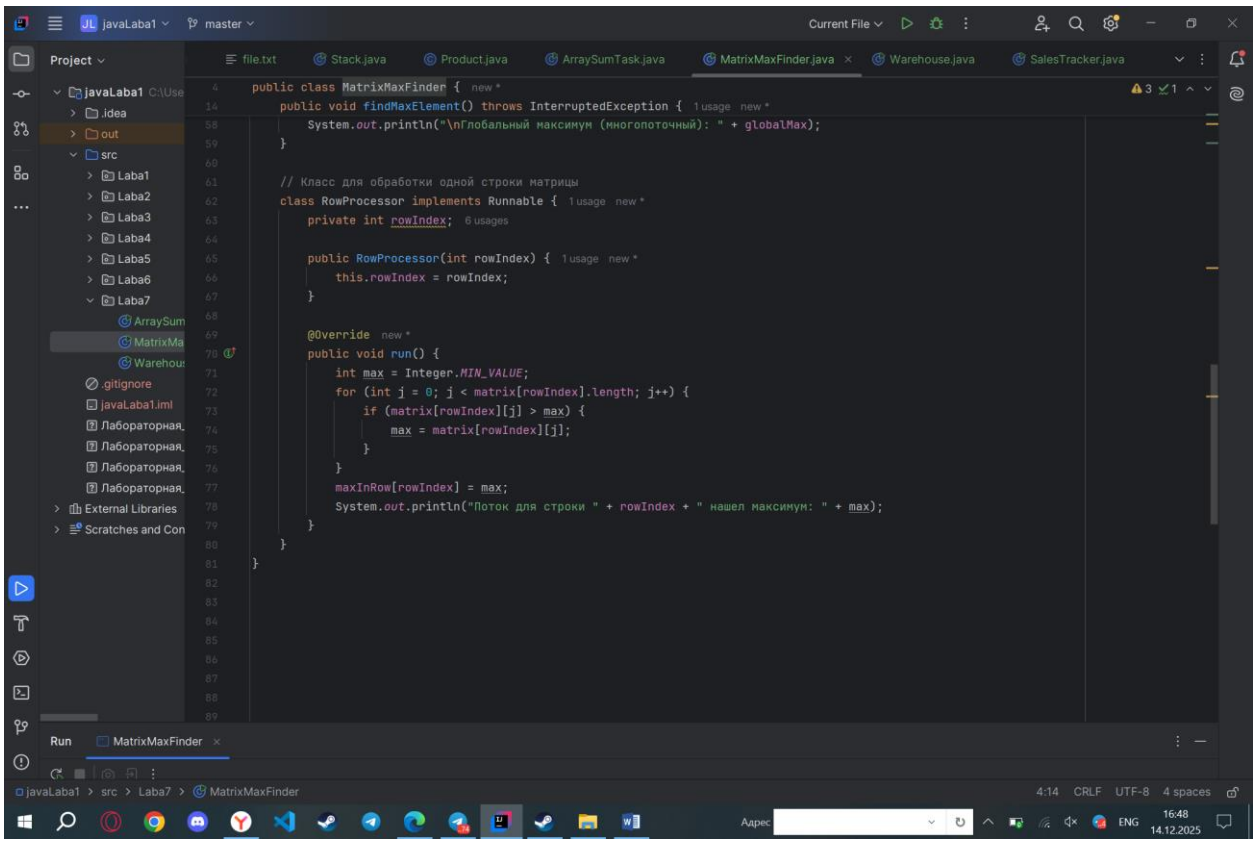
Задание 2



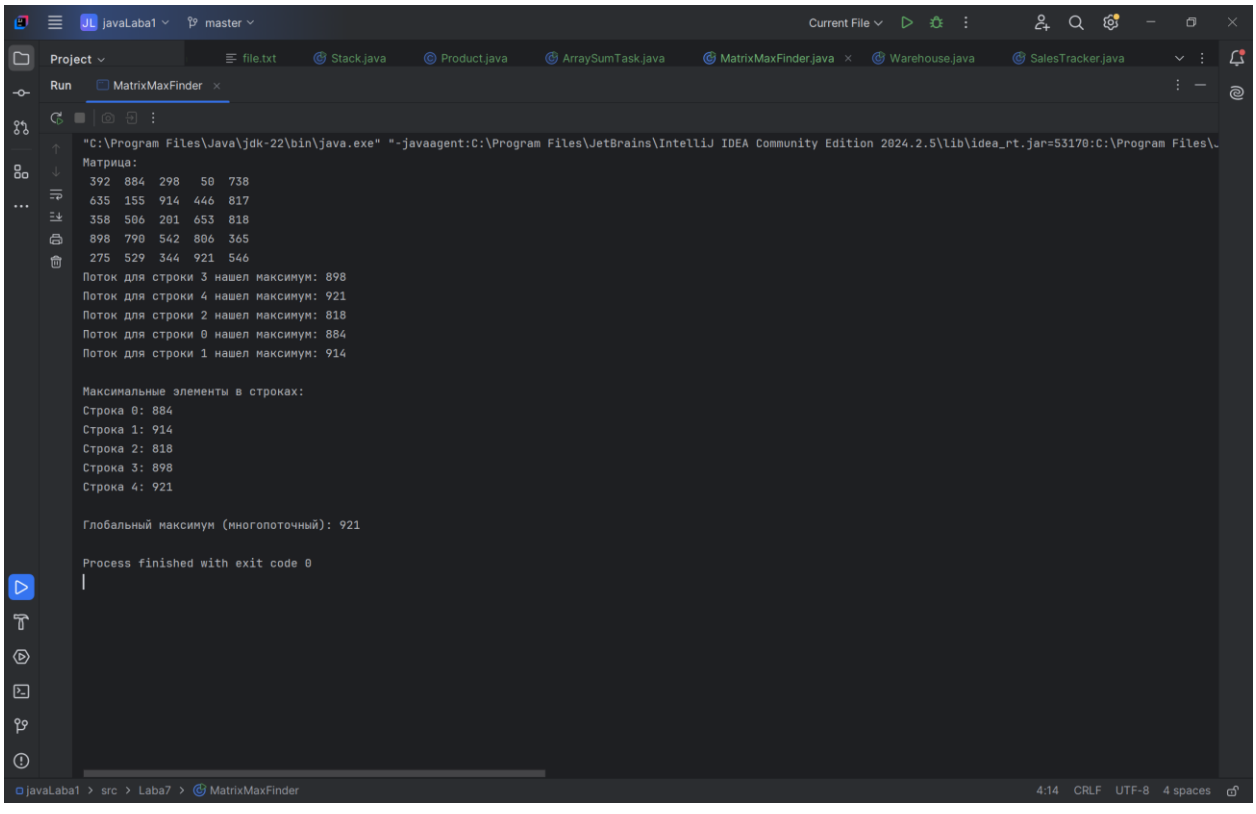

```java
public class MatrixMaxFinder {
    private int[][] matrix;
    private int maxInRow[];
    private int globalMax = Integer.MIN_VALUE;

    public static void main(String[] args) throws InterruptedException {
        MatrixMaxFinder finder = new MatrixMaxFinder();
        finder.findMaxElement();
    }

    public void findMaxElement() throws InterruptedException {
        int rows = 5;
        int cols = 5;
        matrix = new int[rows][cols];
        maxInRow = new int[rows];

        Random random = new Random();

        // Заполняем матрицу случайными числами
        System.out.println("Матрица:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = random.nextInt( bound: 1000);
                System.out.printf("%4d ", matrix[i][j]);
            }
            System.out.println();
        }

        // Создаем массив потоков (по одному на строку)
        Thread[] threads = new Thread[rows];

        for (int i = 0; i < rows; i++) {
            int rowIndex = i;
            threads[i] = new Thread(new RowProcessor(rowIndex));
```



```java
public class MatrixMaxFinder {
    public void findMaxElement() throws InterruptedException {
            threads[i].start();
        }

        // Ждем завершения всех потоков
        for (Thread thread : threads) {
            thread.join();
        }

        // Находим глобальный максимум в главном потоке
        for (int i = 0; i < rows; i++) {
            if (maxInRow[i] > globalMax) {
                globalMax = maxInRow[i];
            }
        }


        System.out.println("\nМаксимальные элементы в строках:");
        for (int i = 0; i < rows; i++) {
            System.out.println("Строка " + i + ": " + maxInRow[i]);
        }
        System.out.println("\nГлобальный максимум (многопоточный): " + globalMax);
    }

    // Класс для обработки одной строки матрицы
    class RowProcessor implements Runnable {
        private int rowIndex;

        public RowProcessor(int rowIndex) {
            this.rowIndex = rowIndex;
        }

        @Override
        public void run() {
```

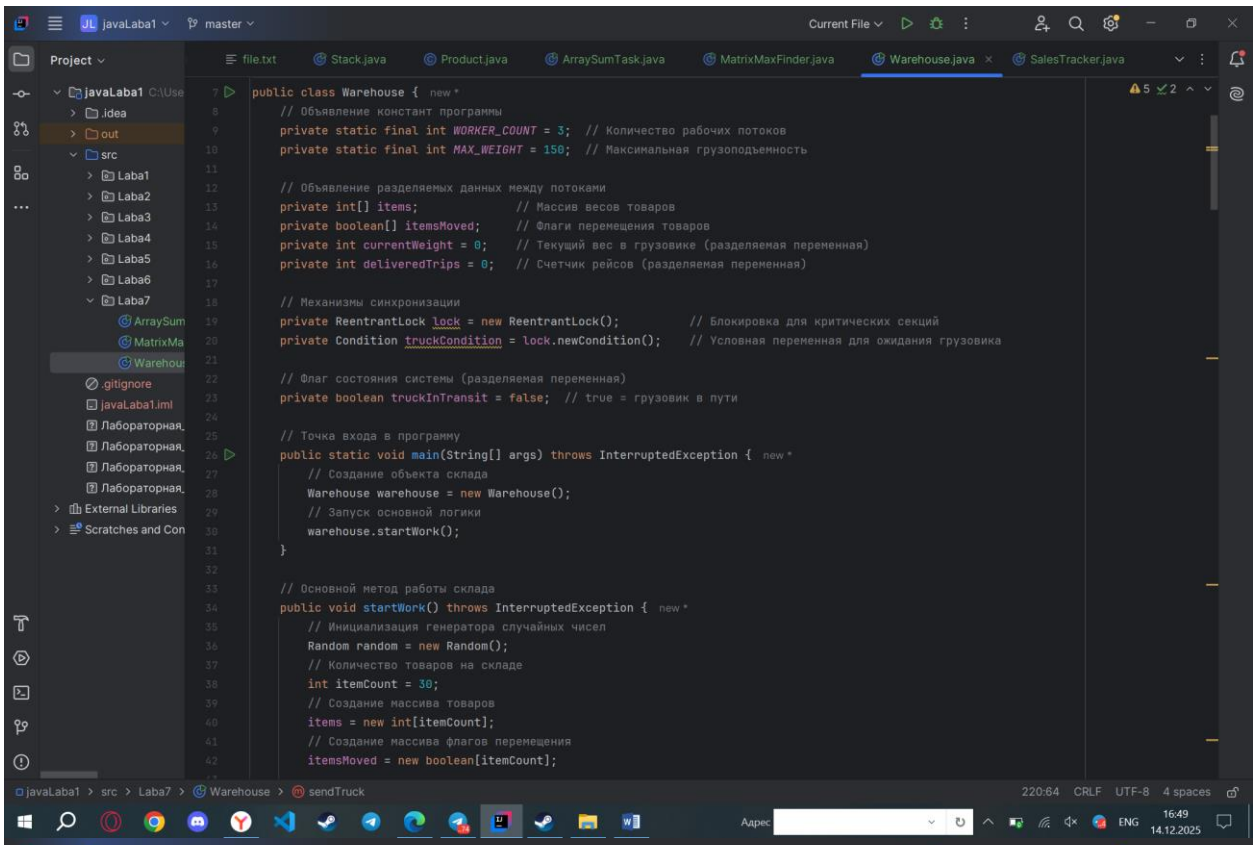


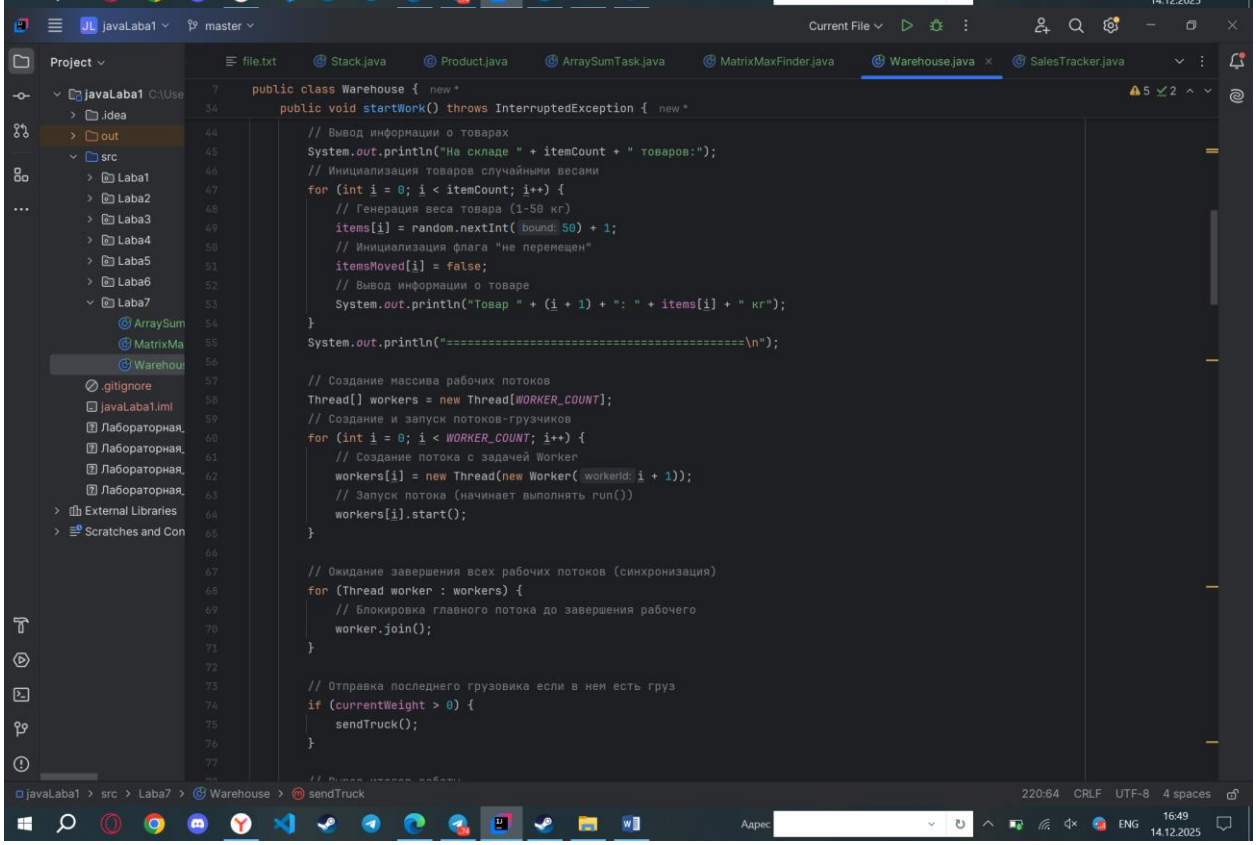
```
public class MatrixMaxFinder {
    public void findMaxElement() throws InterruptedException {
        System.out.println("\nГлобальный максимум (многопоточный): " + globalMax);
    }

    // Класс для обработки одной строки матрицы
    class RowProcessor implements Runnable {
        private int rowIndex;

        public RowProcessor(int rowIndex) {
            this.rowIndex = rowIndex;
        }

        @Override
        public void run() {
            int max = Integer.MIN_VALUE;
            for (int j = 0; j < matrix[rowIndex].length; j++) {
                if (matrix[rowIndex][j] > max) {
                    max = matrix[rowIndex][j];
                }
            }
            maxInRow[rowIndex] = max;
            System.out.println("Поток для строки " + rowIndex + " нашел максимум: " + max);
        }
    }
}
```

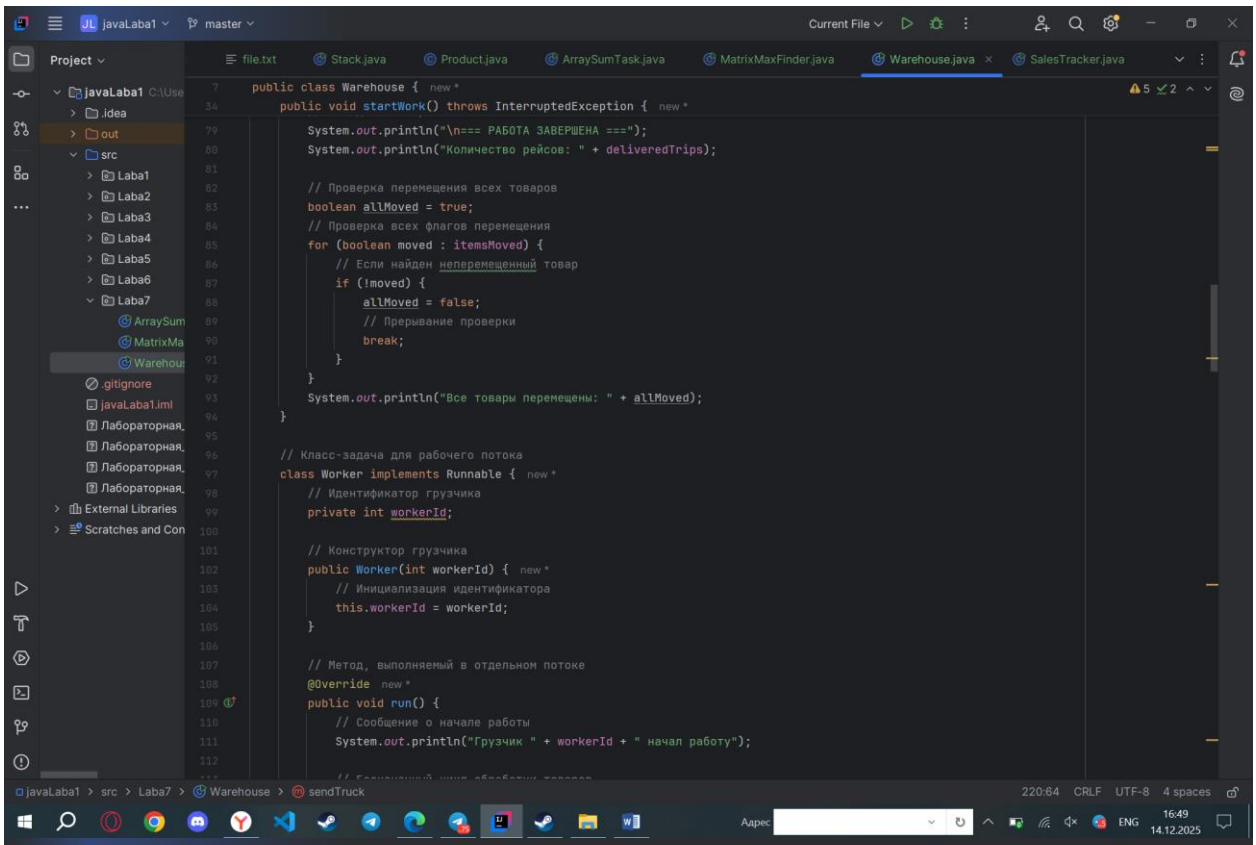


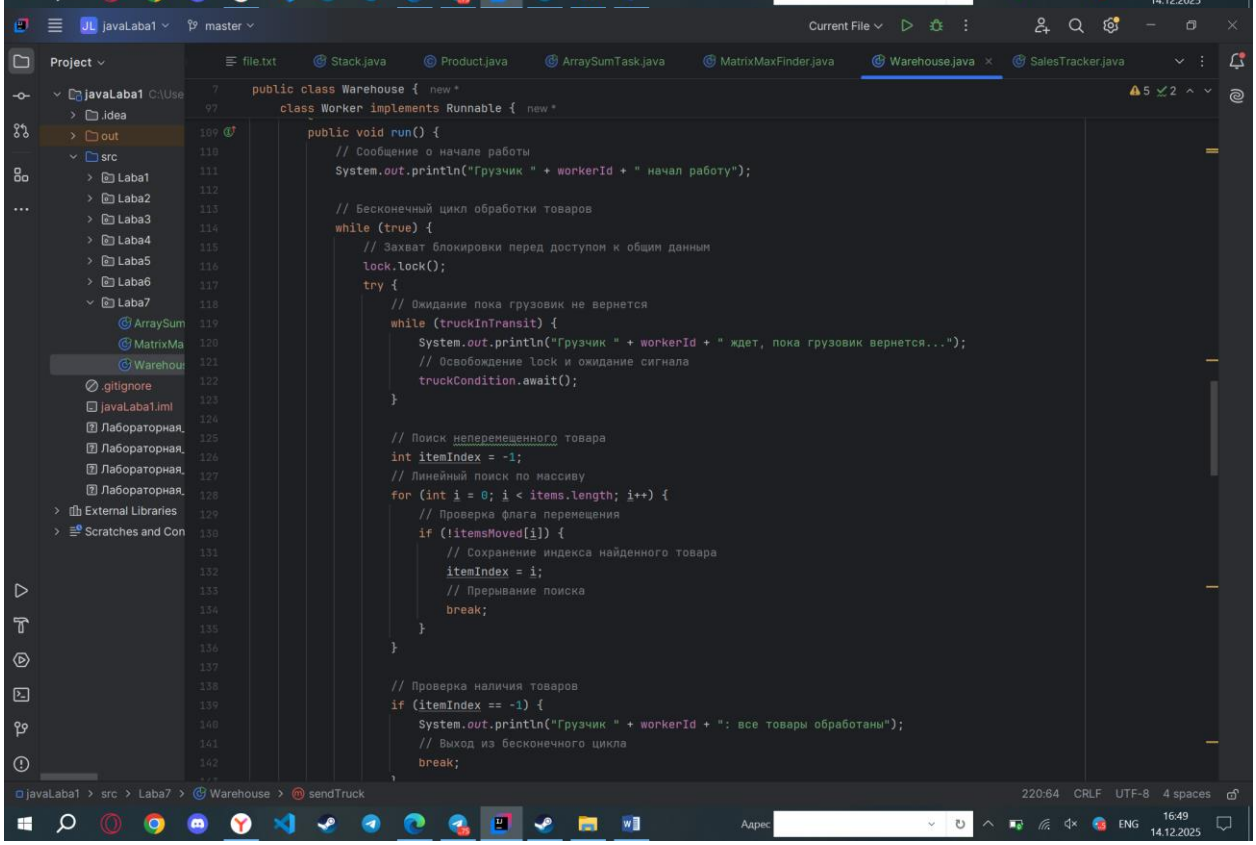
```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=53170:C:\Program Files\
Матрица:
 392  884  298   50  738
 635  155  914  446  817
 358  506  201  653  818
 898  790  542  806  365
 275  529  344  921  546
Поток для строки 3 нашел максимум: 898
Поток для строки 4 нашел максимум: 921
Поток для строки 2 нашел максимум: 818
Поток для строки 0 нашел максимум: 884
Поток для строки 1 нашел максимум: 914

Максимальные элементы в строках:
Строка 0: 884
Строка 1: 914
Строка 2: 818
Строка 3: 898
Строка 4: 921

Глобальный максимум (многопоточный): 921

Process finished with exit code 0
```

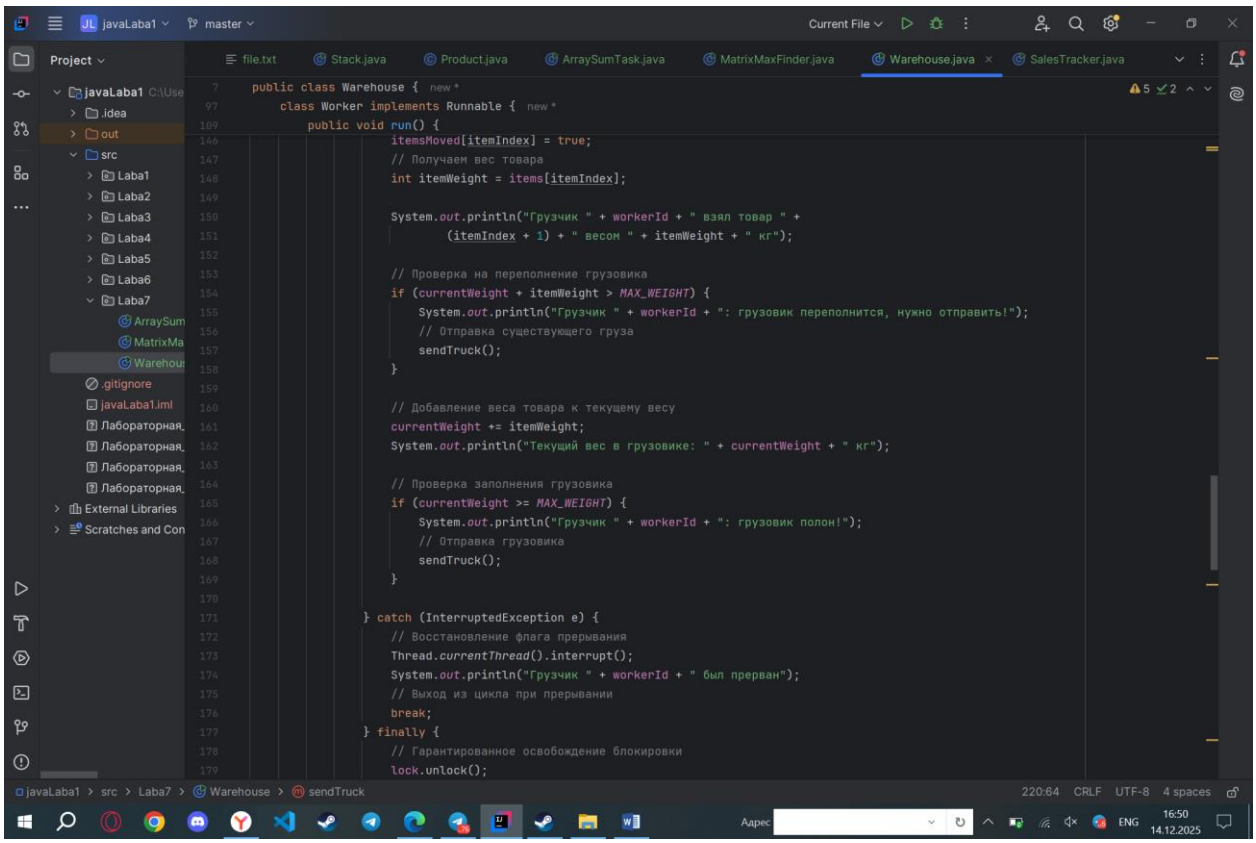
Задание 3


javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
javaLaba1 C:\Use
.idea
out
src
Laba1
Laba2
Laba3
Laba4
Laba5
Laba6
Laba7
ArraySum
MatrixMa
Warehou
.gitignore
javaLaba1.iml
Лабораторная_
Лабораторная_
Лабораторная_
Лабораторная_
External Libraries
Scratches and Con
public class Warehouse { new *
// Объявление констант программы
private static final int WORKER_COUNT = 3; // Количество рабочих потоков
private static final int MAX_WEIGHT = 150; // Максимальная грузоподъемность
// Объявление разделяемых данных между потоками
private int[] items; // Массив весов товаров
private boolean[] itemsMoved; // Флаги перемещения товаров
private int currentWeight = 0; // Текущий вес в грузовике (разделяемая переменная)
private int deliveredTrips = 0; // Счетчик рейсов (разделяемая переменная)
// Механизмы синхронизации
private ReentrantLock lock = new ReentrantLock(); // Блокировка для критических секций
private Condition truckCondition = lock.newCondition(); // Условная переменная для ожидания грузовика
// Флаг состояния системы (разделяемая переменная)
private boolean truckInTransit = false; // true = грузовик в пути
// Точка входа в программу
public static void main(String[] args) throws InterruptedException { new *
// Создание объекта склада
Warehouse warehouse = new Warehouse();
// Запуск основной логики
warehouse.startWork();
}
// Основной метод работы склада
public void startWork() throws InterruptedException { new *
// Инициализация генератора случайных чисел
Random random = new Random();
// Количество товаров на складе
int itemCount = 30;
// Создание массива товаров
items = new int[itemCount];
// Создание массива флагов перемещения
itemsMoved = new boolean[itemCount];
javaLaba1 > src > Laba7 > Warehouse > sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:49
14.12.2025

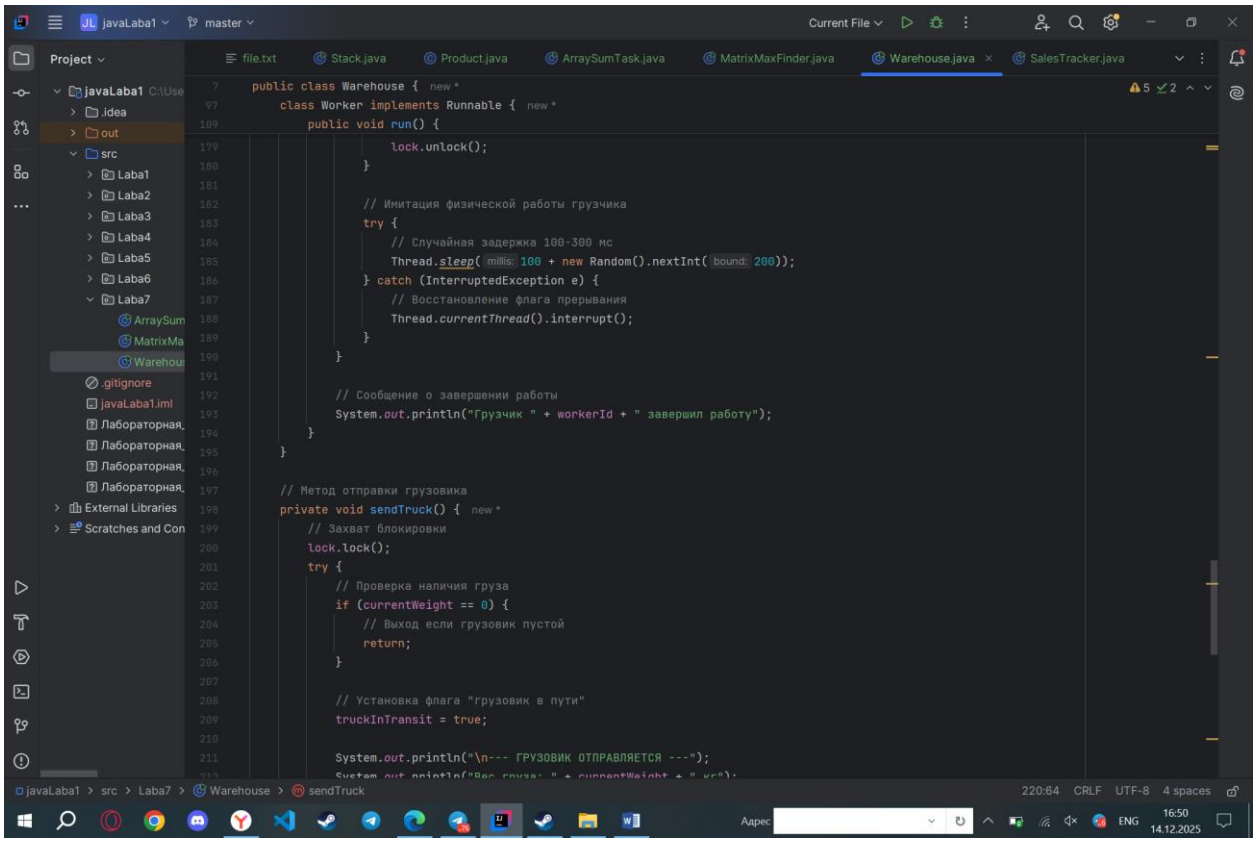

javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
public class Warehouse { new *
public void startWork() throws InterruptedException { new *
// Вывод информации о товарах
System.out.println("На складе " + itemCount + " товаров:");
// Инициализация товаров случайными весами
for (int i = 0; i < itemCount; i++) {
// Генерация веса товара (1-50 кг)
items[i] = random.nextInt(bound: 50) + 1;
// Инициализация флага "не перемещен"
itemsMoved[i] = false;
// Вывод информации о товаре
System.out.println("Товар " + (i + 1) + ": " + items[i] + " кг");
}
System.out.println("==========\n");
// Создание массива рабочих потоков
Thread[] workers = new Thread[WORKER_COUNT];
// Создание и запуск потоков-грузчиков
for (int i = 0; i < WORKER_COUNT; i++) {
// Создание потока с задачей Worker
workers[i] = new Thread(new Worker(workerId: i + 1));
// Запуск потока (начинает выполнять run())
workers[i].start();
}
// Ожидание завершения всех рабочих потоков (синхронизация)
for (Thread worker : workers) {
// Блокировка главного потока до завершения рабочего
worker.join();
}
// Отправка последнего грузовика если в нем есть груз
if (currentWeight > 0) {
sendTruck();
}
javaLaba1 > src > Laba7 > Warehouse > sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:49
14.12.2025


javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
public class Warehouse {
public void startWork() throws InterruptedException {
System.out.println("\n=== РАБОТА ЗАВЕРШЕНА ===");
System.out.println("Количество рейсов: " + deliveredTrips);
// Проверка перемещения всех товаров
boolean allMoved = true;
// Проверка всех флагов перемещения
for (boolean moved : itemsMoved) {
// Если найден неперемещенный товар
if (!moved) {
allMoved = false;
// Прерывание проверки
break;
}
}
System.out.println("Все товары перемещены: " + allMoved);
}
// Класс-задача для рабочего потока
class Worker implements Runnable {
// Идентификатор грузчика
private int workerId;
// Конструктор грузчика
public Worker(int workerId) {
// Инициализация идентификатора
this.workerId = workerId;
}
// Метод, выполняемый в отдельном потоке
@Override
public void run() {
// Сообщение о начале работы
System.out.println("Грузчик " + workerId + " начал работу");
javaLaba1 › src › Laba7 › Warehouse › sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:49
14.12.2025

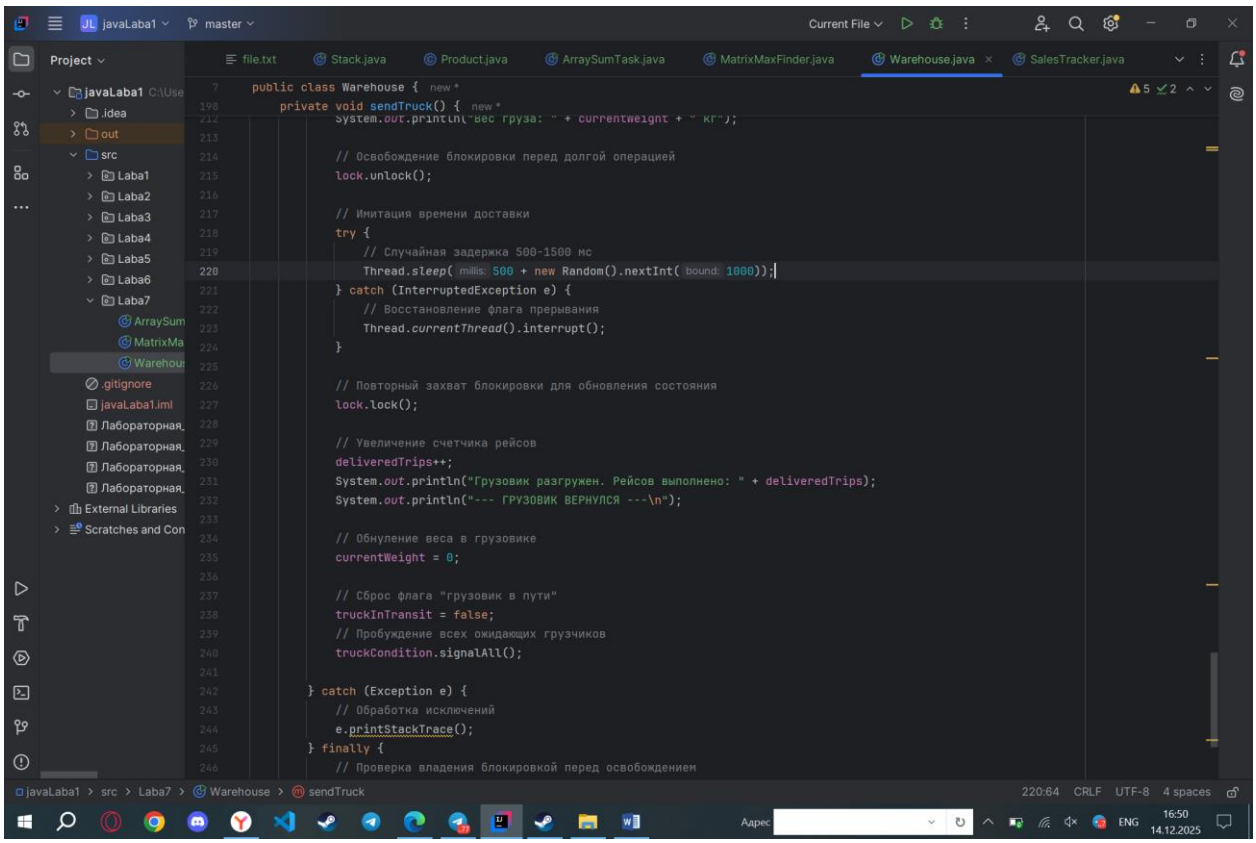

javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
public class Warehouse {
class Worker implements Runnable {
public void run() {
// Сообщение о начале работы
System.out.println("Грузчик " + workerId + " начал работу");
// Бесконечный цикл обработки товаров
while (true) {
// Захват блокировки перед доступом к общим данным
lock.lock();
try {
// Ожидание пока грузовик не вернется
while (truckInTransit) {
System.out.println("Грузчик " + workerId + " ждет, пока грузовик вернется...");
// Освобождение lock и ожидание сигнала
truckCondition.await();
}
// Поиск неперемещенного товара
int itemIndex = -1;
// Линейный поиск по массиву
for (int i = 0; i < items.length; i++) {
// Проверка флага перемещения
if (!itemsMoved[i]) {
// Сохранение индекса найденного товара
itemIndex = i;
// Прерывание поиска
break;
}
}
// Проверка наличия товаров
if (itemIndex == -1) {
System.out.println("Грузчик " + workerId + ": все товары обработаны");
// Выход из бесконечного цикла
break;
javaLaba1 › src › Laba7 › Warehouse › sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:49
14.12.2025



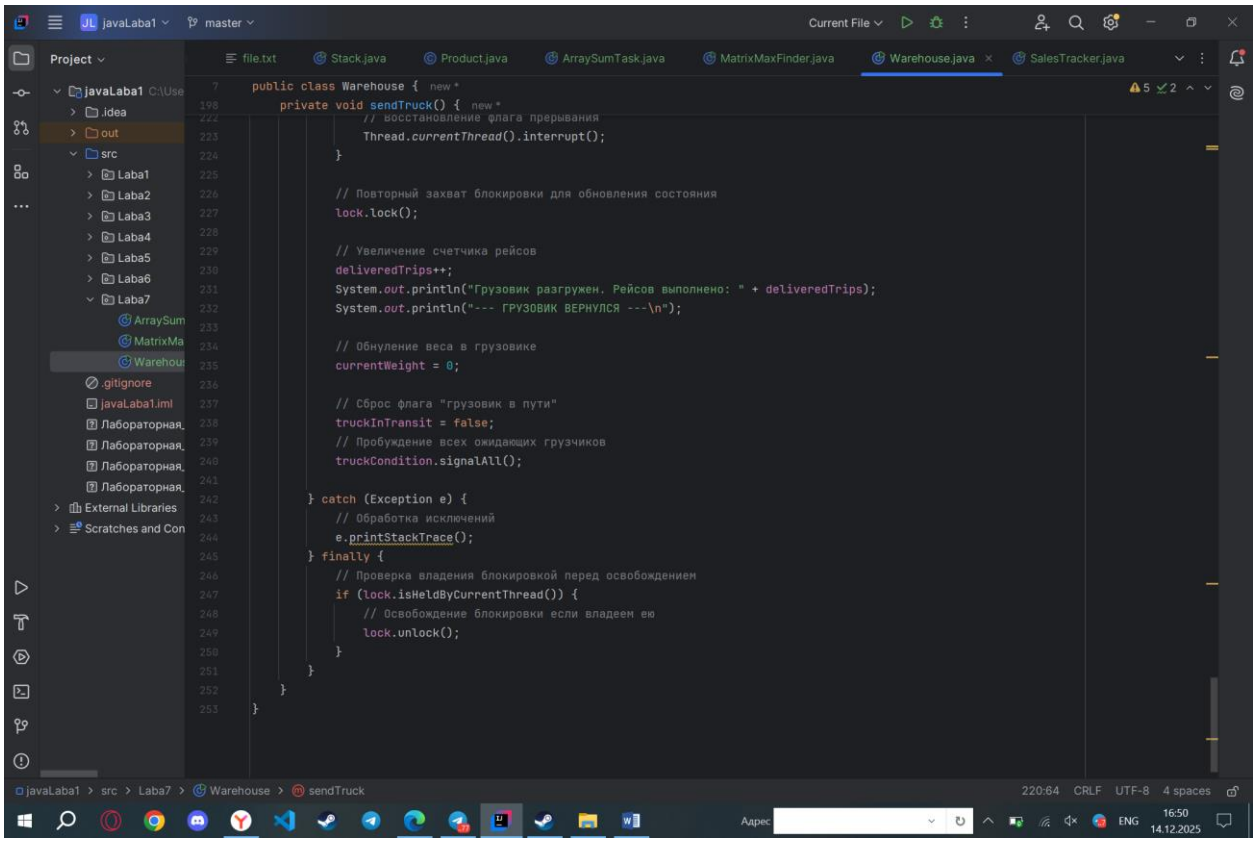
```
public class Warehouse {
    class Worker implements Runnable {
        public void run() {
                    itemsMoved[itemIndex] = true;
                    // Получаем вес товара
                    int itemWeight = items[itemIndex];

                    System.out.println("Грузчик " + workerId + " взял товар " +
                            (itemIndex + 1) + " весом " + itemWeight + " кг");

                    // Проверка на переполнение грузовика
                    if (currentWeight + itemWeight > MAX_WEIGHT) {
                        System.out.println("Грузчик " + workerId + ": грузовик переполнится, нужно отправить!");
                        // Отправка существующего груза
                        sendTruck();
                    }

                    // Добавление веса товара к текущему весу
                    currentWeight += itemWeight;
                    System.out.println("Текущий вес в грузовике: " + currentWeight + " кг");

                    // Проверка заполнения грузовика
                    if (currentWeight >= MAX_WEIGHT) {
                        System.out.println("Грузчик " + workerId + ": грузовик полон!");
                        // Отправка грузовика
                        sendTruck();
                    }

                } catch (InterruptedException e) {
                    // Восстановление флага прерывания
                    Thread.currentThread().interrupt();
                    System.out.println("Грузчик " + workerId + " был прерван");
                    // Выход из цикла при прерывании
                    break;
                } finally {
                    // Гарантированное освобождение блокировки
                    lock.unlock();
```



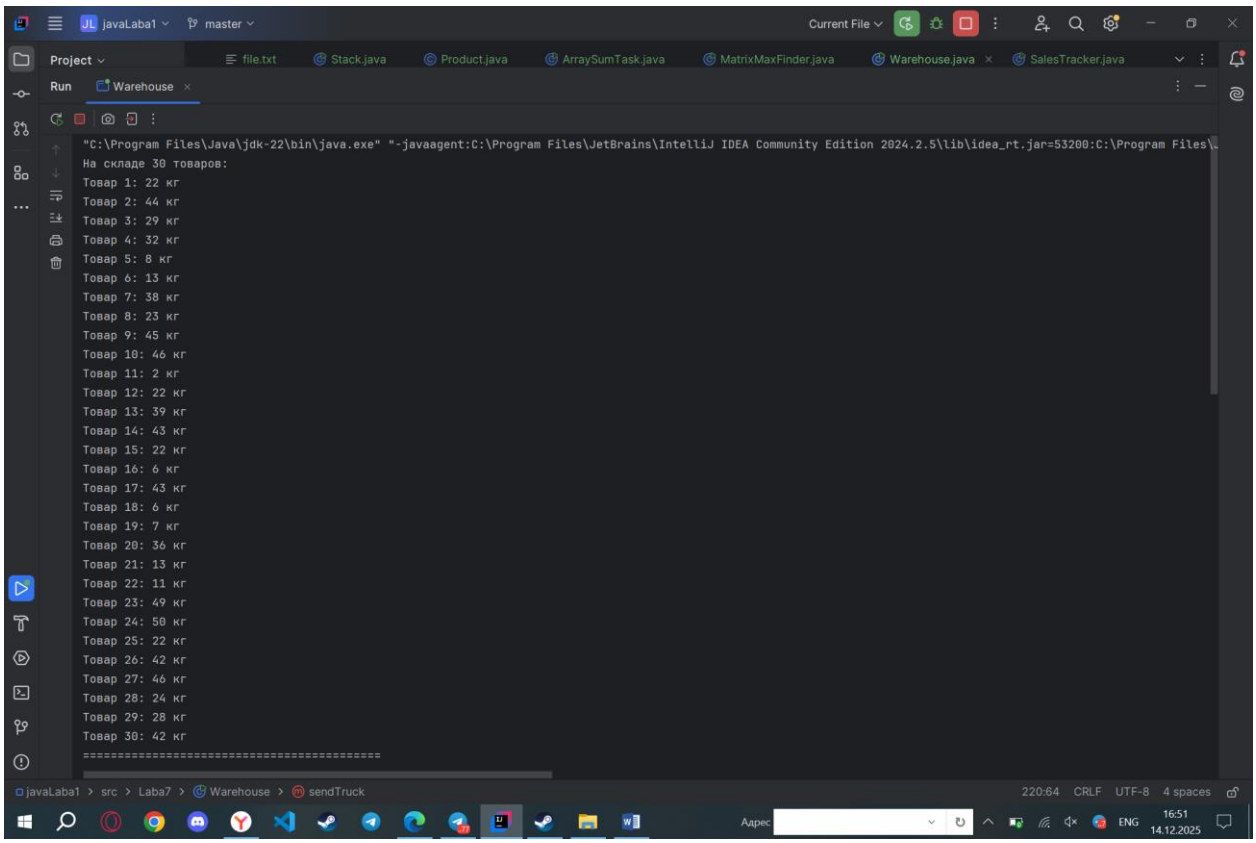
```
                    lock.unlock();
                }

                // Имитация физической работы грузчика
                try {
                    // Случайная задержка 100-300 мс
                    Thread.sleep( millis: 100 + new Random().nextInt( bound: 200));
                } catch (InterruptedException e) {
                    // Восстановление флага прерывания
                    Thread.currentThread().interrupt();
                }
            }

            // Сообщение о завершении работы
            System.out.println("Грузчик " + workerId + " завершил работу");
        }
    }

    // Метод отправки грузовика
    private void sendTruck() {
        // Захват блокировки
        lock.lock();
        try {
            // Проверка наличия груза
            if (currentWeight == 0) {
                // Выход если грузовик пустой
                return;
            }

            // Установка флага "грузовик в пути"
            truckInTransit = true;

            System.out.println("\n--- ГРУЗОВИК ОТПРАВЛЯЕТСЯ ---");
```

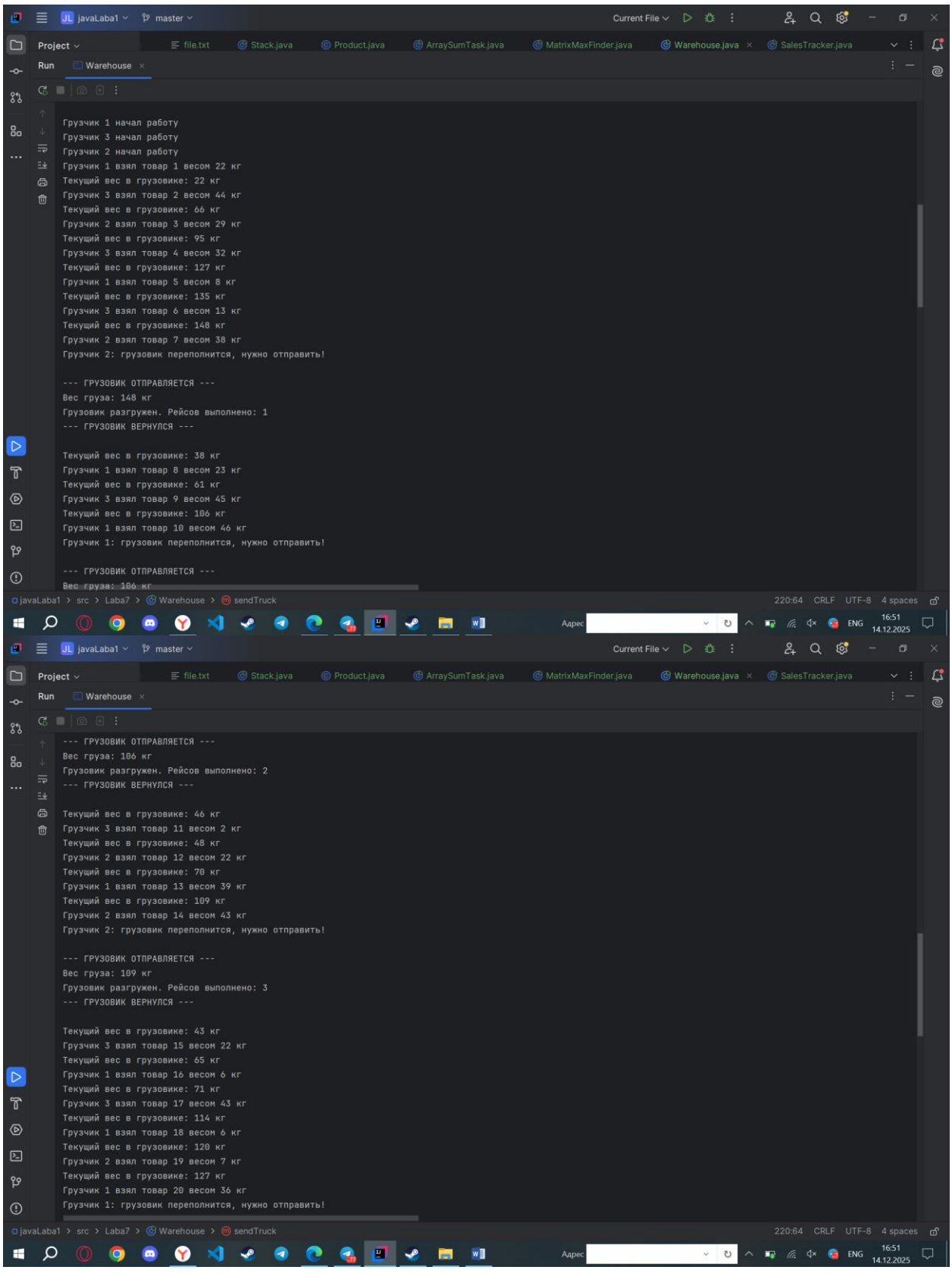


```
public class Warehouse {  new *
    private void sendTruck() {  new *
                System.out.println("Вес груза: " + currentWeight + " кг");

                // Освобождение блокировки перед долгой операцией
                lock.unlock();

                // Имитация времени доставки
                try {
                    // Случайная задержка 500-1500 мс
                    Thread.sleep( millis: 500 + new Random().nextInt( bound: 1000));
                } catch (InterruptedException e) {
                    // Восстановление флага прерывания
                    Thread.currentThread().interrupt();
                }

                // Повторный захват блокировки для обновления состояния
                lock.lock();

                // Увеличение счетчика рейсов
                deliveredTrips++;
                System.out.println("Грузовик разгружен. Рейсов выполнено: " + deliveredTrips);
                System.out.println("--- ГРУЗОВИК ВЕРНУЛСЯ ---\n");

                // Обнуление веса в грузовике
                currentWeight = 0;

                // Сброс флага "грузовик в пути"
                truckInTransit = false;
                // Пробуждение всех ожидающих грузчиков
                truckCondition.signalAll();

            } catch (Exception e) {
                // Обработка исключений
                e.printStackTrace();
            } finally {
                // Проверка владения блокировкой перед освобождением
```

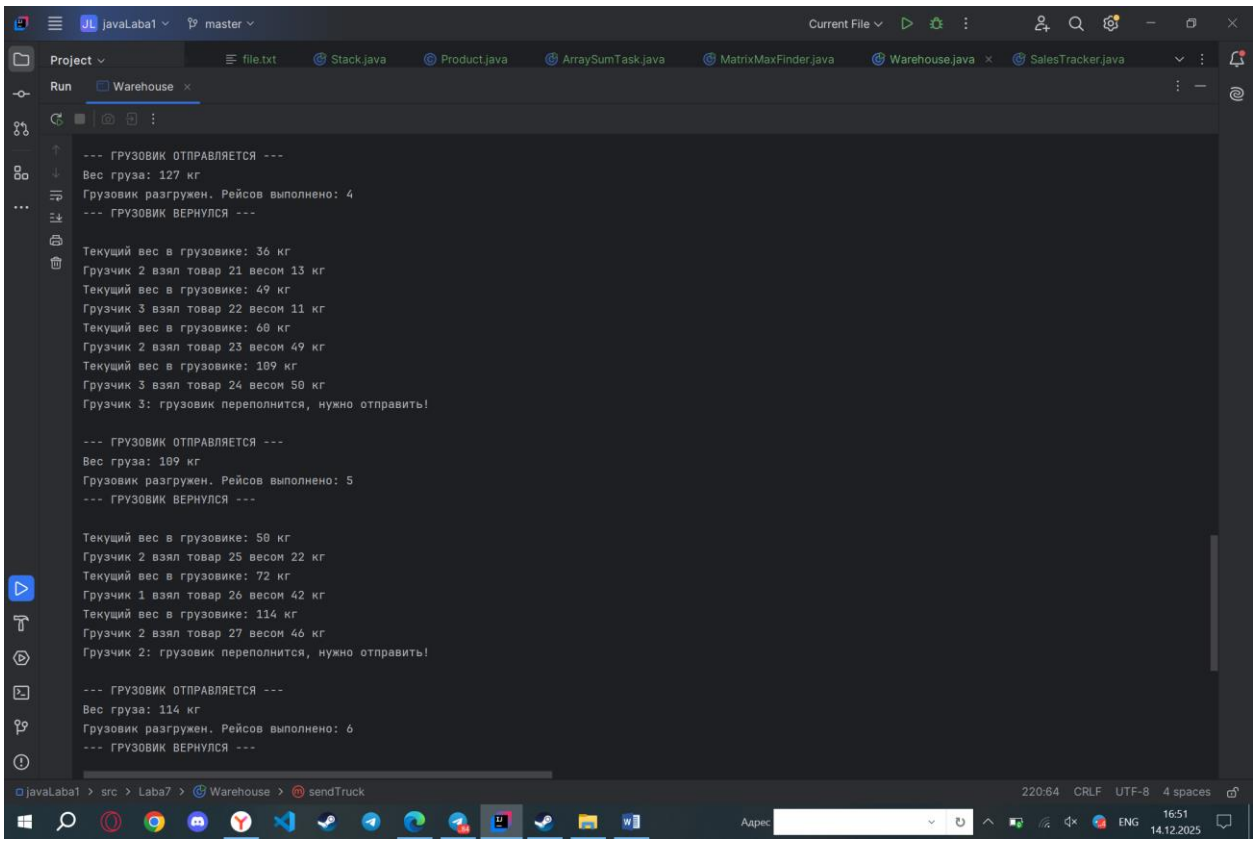


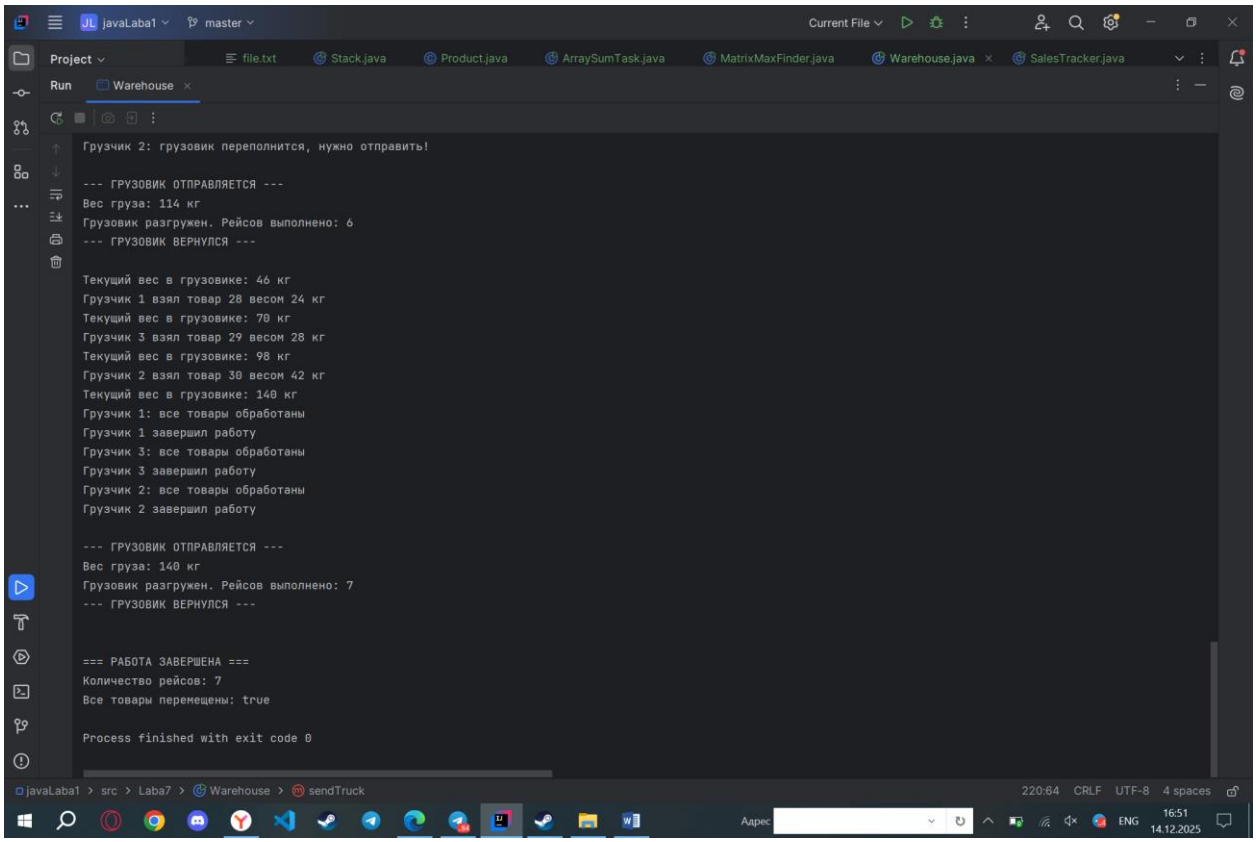
```
public class Warehouse {  new *
    private void sendTruck() {  new *
                    // Восстановление флага прерывания
                    Thread.currentThread().interrupt();
                }

                // Повторный захват блокировки для обновления состояния
                lock.lock();

                // Увеличение счетчика рейсов
                deliveredTrips++;
                System.out.println("Грузовик разгружен. Рейсов выполнено: " + deliveredTrips);
                System.out.println("--- ГРУЗОВИК ВЕРНУЛСЯ ---\n");

                // Обнуление веса в грузовике
                currentWeight = 0;

                // Сброс флага "грузовик в пути"
                truckInTransit = false;
                // Пробуждение всех ожидающих грузчиков
                truckCondition.signalAll();

            } catch (Exception e) {
                // Обработка исключений
                e.printStackTrace();
            } finally {
                // Проверка владения блокировкой перед освобождением
                if (lock.isHeldByCurrentThread()) {
                    // Освобождение блокировки если владеем ею
                    lock.unlock();
                }
            }
        }
    }
```


javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
Run
Warehouse
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=53200:C:\Program Files\
На складе 30 товаров:
Товар 1: 22 кг
Товар 2: 44 кг
Товар 3: 29 кг
Товар 4: 32 кг
Товар 5: 8 кг
Товар 6: 13 кг
Товар 7: 38 кг
Товар 8: 23 кг
Товар 9: 45 кг
Товар 10: 46 кг
Товар 11: 2 кг
Товар 12: 22 кг
Товар 13: 39 кг
Товар 14: 43 кг
Товар 15: 22 кг
Товар 16: 6 кг
Товар 17: 43 кг
Товар 18: 6 кг
Товар 19: 7 кг
Товар 20: 36 кг
Товар 21: 13 кг
Товар 22: 11 кг
Товар 23: 49 кг
Товар 24: 50 кг
Товар 25: 22 кг
Товар 26: 42 кг
Товар 27: 46 кг
Товар 28: 24 кг
Товар 29: 28 кг
Товар 30: 42 кг
==========
javaLaba1 > src > Laba7 > Warehouse > sendTruck
220:64
CRLF
UTF-8
4 spaces
Адрес
ENG
16:51
14.12.2025


javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
Run
Warehouse
Грузчик 1 начал работу
Грузчик 3 начал работу
Грузчик 2 начал работу
Грузчик 1 взял товар 1 весом 22 кг
Текущий вес в грузовике: 22 кг
Грузчик 3 взял товар 2 весом 44 кг
Текущий вес в грузовике: 66 кг
Грузчик 2 взял товар 3 весом 29 кг
Текущий вес в грузовике: 95 кг
Грузчик 3 взял товар 4 весом 32 кг
Текущий вес в грузовике: 127 кг
Грузчик 1 взял товар 5 весом 8 кг
Текущий вес в грузовике: 135 кг
Грузчик 3 взял товар 6 весом 13 кг
Текущий вес в грузовике: 148 кг
Грузчик 2 взял товар 7 весом 38 кг
Грузчик 2: грузовик переполнится, нужно отправить!
--- ГРУЗОВИК ОТПРАВЛЯЕТСЯ ---
Вес груза: 148 кг
Грузовик разгружен. Рейсов выполнено: 1
--- ГРУЗОВИК ВЕРНУЛСЯ ---
Текущий вес в грузовике: 38 кг
Грузчик 1 взял товар 8 весом 23 кг
Текущий вес в грузовике: 61 кг
Грузчик 3 взял товар 9 весом 45 кг
Текущий вес в грузовике: 106 кг
Грузчик 1 взял товар 10 весом 46 кг
Грузчик 1: грузовик переполнится, нужно отправить!
--- ГРУЗОВИК ОТПРАВЛЯЕТСЯ ---
Вес груза: 106 кг
javaLaba1 > src > Laba7 > Warehouse > sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:51
14.12.2025

javaLaba1
master
Current File
Project
file.txt
Stack.java
Product.java
ArraySumTask.java
MatrixMaxFinder.java
Warehouse.java
SalesTracker.java
Run
Warehouse
--- ГРУЗОВИК ОТПРАВЛЯЕТСЯ ---
Вес груза: 106 кг
Грузовик разгружен. Рейсов выполнено: 2
--- ГРУЗОВИК ВЕРНУЛСЯ ---
Текущий вес в грузовике: 46 кг
Грузчик 3 взял товар 11 весом 2 кг
Текущий вес в грузовике: 48 кг
Грузчик 2 взял товар 12 весом 22 кг
Текущий вес в грузовике: 70 кг
Грузчик 1 взял товар 13 весом 39 кг
Текущий вес в грузовике: 109 кг
Грузчик 2 взял товар 14 весом 43 кг
Грузчик 2: грузовик переполнится, нужно отправить!
--- ГРУЗОВИК ОТПРАВЛЯЕТСЯ ---
Вес груза: 109 кг
Грузовик разгружен. Рейсов выполнено: 3
--- ГРУЗОВИК ВЕРНУЛСЯ ---
Текущий вес в грузовике: 43 кг
Грузчик 3 взял товар 15 весом 22 кг
Текущий вес в грузовике: 65 кг
Грузчик 1 взял товар 16 весом 6 кг
Текущий вес в грузовике: 71 кг
Грузчик 3 взял товар 17 весом 43 кг
Текущий вес в грузовике: 114 кг
Грузчик 1 взял товар 18 весом 6 кг
Текущий вес в грузовике: 120 кг
Грузчик 2 взял товар 19 весом 7 кг
Текущий вес в грузовике: 127 кг
Грузчик 1 взял товар 20 весом 36 кг
Грузчик 1: грузовик переполнится, нужно отправить!
javaLaba1 > src > Laba7 > Warehouse > sendTruck
220:64 CRLF UTF-8 4 spaces
Адрес
ENG
16:51
14.12.2025





Заключение

Вывод:

В ходе выполнения лабораторной работы были изучены основы многопоточного программирования в Java. Были рассмотрены различные подходы к созданию и

управлению потоками, механизмы синхронизации доступа к общим ресурсам, а также способы предотвращения типичных проблем многопоточности, таких как гонки данных и взаимные блокировки. Практическая часть работы включала реализацию трех задач, демонстрирующих применение полученных знаний: вычисление суммы элементов массива, поиск максимального элемента в матрице и симуляцию работы склада с грузчиками. Особое внимание было уделено использованию современных механизмов синхронизации из пакета java.util.concurrent, включая ReentrantLock и Condition. Работа показала важность правильного проектирования многопоточных приложений для обеспечения корректности работы и эффективного использования ресурсов системы. Полученные навыки позволяют создавать надежные и производительные многопоточные приложения, что является необходимым требованием в современных условиях, когда многопоточность становится стандартом для разработки высоконагруженных систем.

Ответы на контрольные вопросы:

1. Как реализуется многопоточность в Java?
Многопоточность в Java реализуется с помощью класса Thread и интерфейса Runnable. Основные способы создания потоков: наследование от класса Thread (переопределение метода run()) или реализация интерфейса Runnable (также переопределение run()) с последующей передачей экземпляра в конструктор Thread. Поток запускается методом start(). Для управления потоками и синхронизации используются механизмы synchronized, wait/notify, а также классы из пакета java.util.concurrent (Lock, Semaphore, CountDownLatch и др.).

2. Что такое поток?
Поток (thread) - это наименьшая единица выполнения в процессе. Каждый поток имеет собственный стек вызовов и контекст выполнения, но разделяет память и ресурсы процесса с другими потоками. Потоки позволяют выполнять несколько задач параллельно в рамках одного приложения, что повышает эффективность использования ресурсов процессора.

3. Для чего нужно ключевое слово synchronized?
Ключевое слово synchronized используется для синхронизации доступа к общим ресурсам в многопоточной среде. Оно гарантирует, что только один поток может выполнять синхронизированный блок кода или метод в данный момент времени. Это предотвращает race condition (гонку данных) и обеспечивает корректность работы с разделяемыми данными.

4. Для чего нужно ключевое слово volatile?
Ключевое слово volatile гарантирует видимость изменений переменной для всех потоков. Когда поток изменяет volatile-переменную, это изменение сразу становится видимым другим потокам. Без volatile изменения могут оставаться в кэше процессора и не быть видимыми другим потокам. Однако volatile не обеспечивает атомарность составных операций.

5. Зачем нужно синхронизировать потоки?
Синхронизация потоков необходима для обеспечения корректной работы с общими ресурсами в многопоточной среде. Без синхронизации может возникнуть race condition, когда несколько потоков одновременно изменяют общие данные, приводя к непредсказуемым результатам. Синхронизация также предотвращает deadlock, livelock и обеспечивает последовательный доступ к критическим секциям.

6. Какие есть способы синхронизации потоков?
Основные способы синхронизации: использование ключевого слова synchronized (для методов и блоков), методы wait(), notify(), notifyAll() класса Object, классы из пакета java.util.concurrent (ReentrantLock, Semaphore, CountDownLatch, CyclicBarrier), атомарные классы (AtomicInteger, AtomicBoolean и др.), volatile переменные, коллекции из пакета java.util.concurrent (ConcurrentHashMap, CopyOnWriteArrayList).

7. В чем разница между Thread и Runnable?
Thread - это класс, представляющий поток выполнения. Runnable - это интерфейс, содержащий единственный метод run(). Основные различия: при наследовании от Thread нельзя наследоваться от других классов (ограничение единого наследования Java), а реализация Runnable позволяет наследовать другие классы; Runnable можно передавать в пулы потоков (ExecutorService); Thread инкапсулирует и поведение потока, и задачу, а Runnable разделяет задачу и механизм выполнения.

8. Какие состояния может иметь поток? Опишите жизненный цикл потока.
Поток в Java может находиться в одном из шести состояний: NEW (создан, но не запущен), RUNNABLE (выполняется или готов к выполнению), BLOCKED (блокирован в ожидании монитора), WAITING (ожидает другого потока без таймаута), TIMED_WAITING (ожидание с таймаутом), TERMINATED (завершен). Жизненный цикл: создание (NEW) -> запуск start() (RUNNABLE) -> выполнение -> возможные переходы в BLOCKED/WAITING/TIMED_WAITING -> завершение (TERMINATED).

9. Что такое daemon-поток? Как его создать?
Daemon-поток - это фоновый поток, который не препятствует завершению работы JVM. Когда все пользовательские (не-daemon) потоки завершаются, JVM автоматически завершает все daemon-потоки. Создается с помощью метода setDaemon(true) перед запуском потока (start()). Daemon-потоки обычно используются для фоновых задач, таких как сборка мусора, мониторинг и т.д.

10. Как принудительно остановить поток?
Метод stop() считается устаревшим и опасным, так как он принудительно завершает поток, не обеспечивая корректного освобождения ресурсов. Рекомендуемый способ - использовать флаг завершения (например, volatile boolean flag) и периодически проверять его в методе run(). Также можно использовать interrupt() для прерывания потока, что вызовет InterruptedException в методах, которые его бросают (sleep(), wait(), join()).

11. Как работает метод join()? Для чего он используется?
Метод join() заставляет текущий поток ждать завершения потока, для которого вызван join(). Если вызвать thread.join(), то текущий поток блокируется до тех пор, пока поток thread не завершит свою работу. Можно указать таймаут: join(millis). Используется для синхронизации потоков, когда нужно дождаться завершения одного потока перед продолжением работы другого.

12. Что такое «гонка данных» (race condition)?
Гонка данных (race condition) - это ситуация в многопоточном программировании, когда несколько потоков одновременно обращаются к общим данным и как минимум один поток изменяет эти данные. Результат выполнения программы становится непредсказуемым и зависит от порядка выполнения потоков, который определяется планировщиком потоков операционной системы.

13. Что такое deadlock? Как его избежать?
Deadlock (взаимная блокировка) - это ситуация, когда два или более потока бесконечно ждут друг друга, освобождения ресурсов. Deadlock возникает при одновременном выполнении четырех условий: взаимное исключение, удержание и ожидание, отсутствие вытеснения, циклическое ожидание. Для избежания deadlock нужно: использовать единый порядок захвата блокировок, применять tryLock() с таймаутом, использовать мониторы более высокого уровня, избегать вложенных блокировок.

14. Что такое wait(), notify() и notifyAll()? В каком классе они объявлены?
Это методы для межпоточной коммуникации, объявленные в классе Object. wait() - переводит текущий поток в состояние ожидания и освобождает монитор; notify() - пробуждает один случайный поток, ожидающий на этом мониторе; notifyAll() - пробуждает все потоки, ожидающие на этом мониторе. Эти методы должны вызываться только из синхронизированного контекста (блока synchronized).

15. Что такое ThreadPool? Какие реализации ExecutorService есть в Java?
ThreadPool (пул потоков) - это набор заранее созданных потоков, готовых к выполнению задач. Позволяет избежать накладных расходов на создание и уничтожение потоков для каждой задачи. Реализации ExecutorService: FixedThreadPool (фиксированное количество потоков), CachedThreadPool (создает новые потоки по мере необходимости), SingleThreadExecutor (один поток), ScheduledThreadPool (для отложенного и периодического выполнения), ForkJoinPool (для задач типа divide-and-conquer).

UитХаб с файлами кода: [NT-005-TN/ITiP_LabWorks_Trukhina]