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:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чеуж А.А.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т. А. Приходько

Краснодар

2025

**Тема**: Потоки Executable.

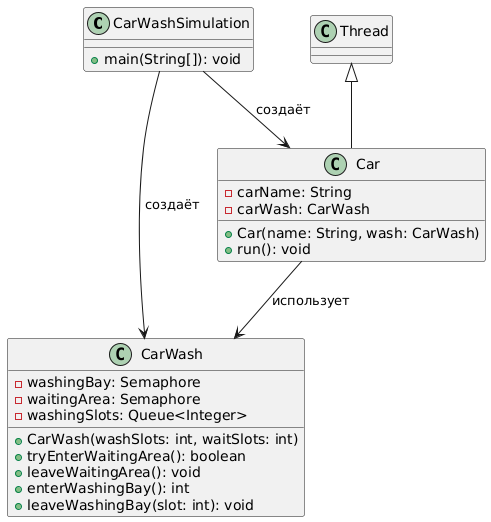
**Цель работы**: Изучить способы организации многопоточной работы приложений на Java.

**Ход работы:**

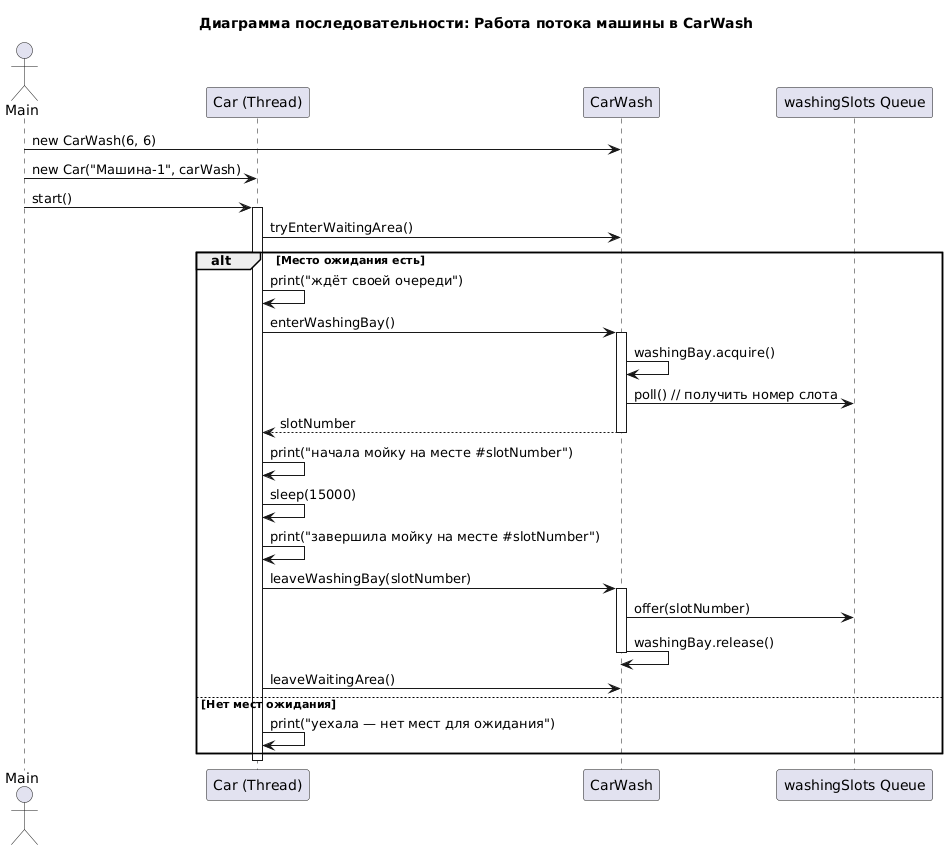
Задание:

На мойку самообслуживания в среднем за час приезжает 10 автомобилей. Мойка оборудована шестью моечным агрегатами, среднее время мойки одного автомобиля — 15 мин. На стоянке есть шесть мест для ожидающих машин. Смоделировать работу мойки, провести анализ ее работы.

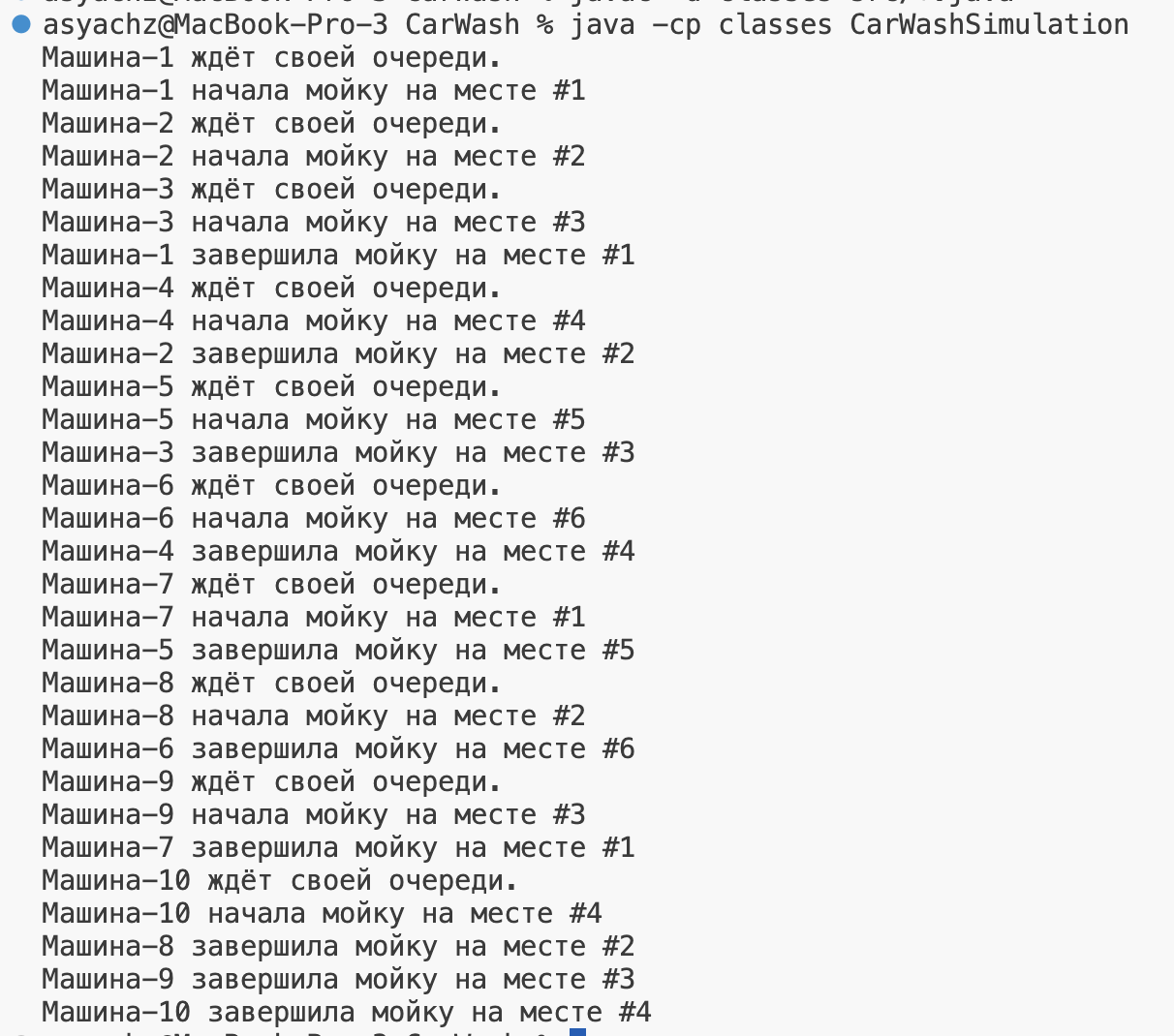
Была построена UML-диаграмма классов:



Для упрощения понимания работы программы была построена диаграмма потоков:



Пример работы программы:



Листнинг программы:

public class CarWashSimulation {

public static void main(String[] args) {

CarWash carWash = new CarWash(6, 6); // 6 мест для мойки и 6 мест парковки

// 10 машин в час => каждые 6 минут по 1 машине

int totalCars = 10;

for (int i = 1; i <= totalCars; i++) {

Car car = new Car("Машина-" + i, carWash);

car.start();

try {

Thread.sleep(2 \* 1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

import java.util.LinkedList;

import java.util.Queue;

import java.util.concurrent.Semaphore; // отвечает за контроль количества потоков, которым разрешён доступ к ресурсу (мойка или ожидание)

public class CarWash {

private final Semaphore washingBay;

private final Semaphore waitingArea;

private final Queue<Integer> washingSlots;

public CarWash(int washingSlotsCount, int waitingSlots) {

this.washingBay = new Semaphore(washingSlotsCount, true);

this.waitingArea = new Semaphore(waitingSlots, true);

this.washingSlots = new LinkedList<>();

for (int i = 1; i <= washingSlotsCount; i++) {

washingSlots.offer(i); // добавляет элемент в конец очереди, если есть место

}

}

public boolean tryEnterWaitingArea() {

return waitingArea.tryAcquire(); // неблокирующий вызов, пытающийся получить разрешение от семафора, возвращает false, если ожидание невозможно

}

public void leaveWaitingArea() {

waitingArea.release(); // освобождает одно разрешение, тем самым разблокирует один из ожидающих потоков, если такой есть

}

public int enterWashingBay() throws InterruptedException {

washingBay.acquire(); // блокирует поток, пока не освободится разрешение, далее поток продолжает выполнение

synchronized (washingSlots) {

return washingSlots.poll(); // удаляет и возвращает первый элемент очереди (забираем место)

}

}

public void leaveWashingBay(int slotNumber) {

synchronized (washingSlots) {

washingSlots.offer(slotNumber); // возвращаем место

}

washingBay.release();

}

}

public class Car extends Thread {

private final String carName;

private final CarWash carWash;

public Car(String carName, CarWash carWash) {

this.carName = carName;

this.carWash = carWash;

}

@Override

public void run() {

if (!carWash.tryEnterWaitingArea()) {

System.out.println(carName + " уехала — нет мест для ожидания.");

return;

}

try {

System.out.println(carName + " ждёт своей очереди.");

int slotNumber = carWash.enterWashingBay();

System.out.println(carName + " начала мойку на месте #" + slotNumber);

Thread.sleep(8 \* 1000); // моделируем мойку

System.out.println(carName + " завершила мойку на месте #" + slotNumber);

carWash.leaveWashingBay(slotNumber);

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

carWash.leaveWaitingArea();

}

}

}

Вывод: В ходе лабораторной работы были изучены методы класса Thread, способы организовать многопоточное выполнение программы на языке программирования Java.