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 №6**

**Дисциплина: Платформо - независимое программирование**

**Тема: «Потоки»**

Работу выполнил(а):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мазуренко А. А.

Направление подготовки: 02.03.02 Фундаментальная информатика

и информационные технологии

Направленность (профиль): Математическое и программное

обеспечение компьютерных технологий

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

Краснодар

2025

**Цель работы:** Изучить приемы и методы организации и управления

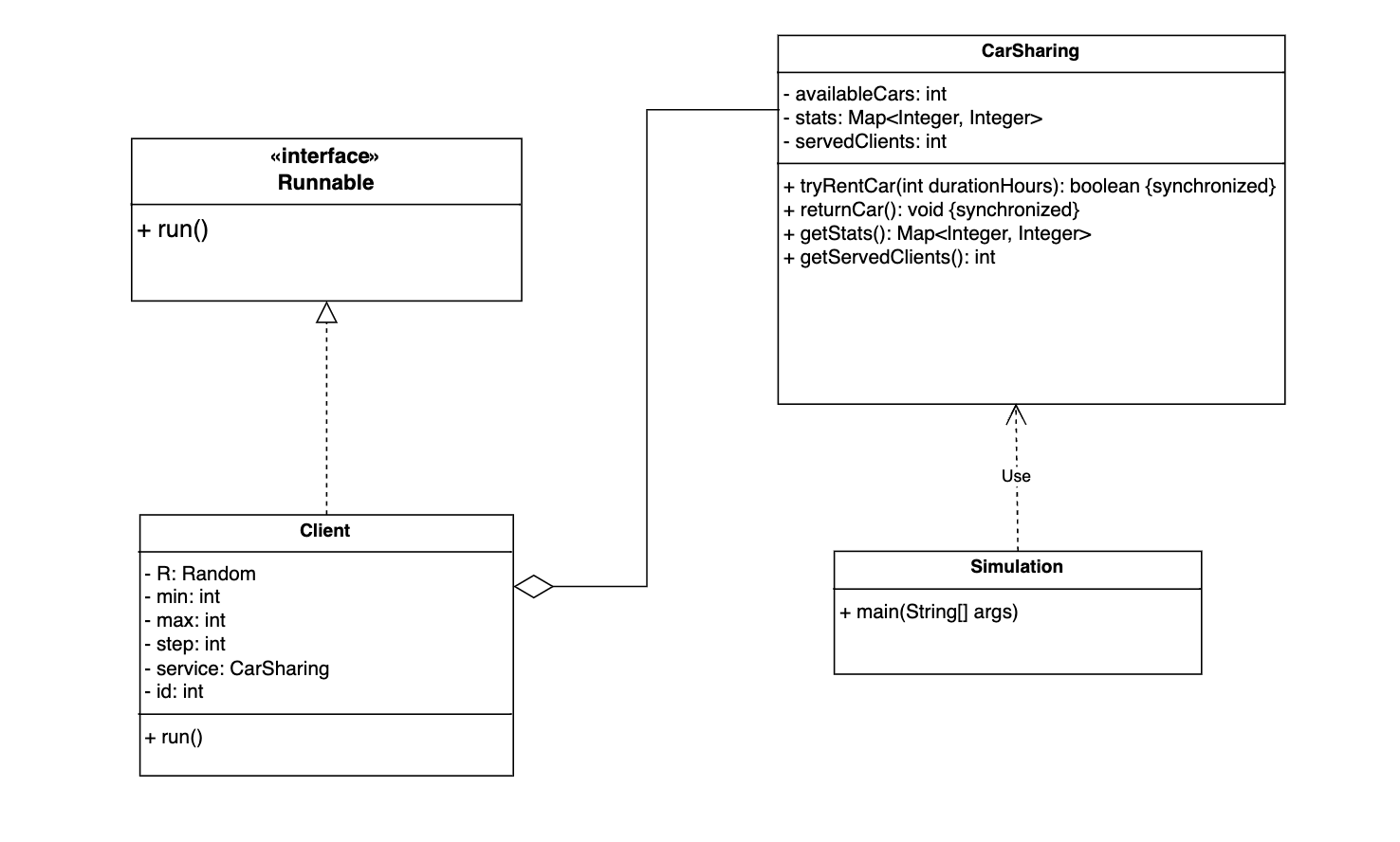
потоками.

**Ход работы:**

**Задача :**

Каршеринг. В службе каршеринга 30 автомобилей. Каждый час приходит клиент - новый поток - и берет машину на срок от 4 до 48 часов (время определяется случайно и время проката всегда четное число). Через оговоренный срок машину возвращают. Подсчитать сколько клиентов воспользовались машинами за сутки. Вывести статистику сколько машин брались на каждый интервал времени: 4, 6, 8… 48 часов.

1. **Диаграмма классов**

****

1. **Листинг программы**

Файл CarSharing.java :

package org.example;

import java.util.HashMap;

import java.util.Map;

public class CarSharing {

private int availableCars;

private final Map<Integer, Integer> stats = new HashMap<>();

private int servedClients = 0;

public CarSharing(int totalCars) {

this.availableCars = totalCars;

}

public synchronized boolean tryRentCar(int durationHours) {

if (availableCars == 0) return false;

availableCars--;

stats.merge(durationHours, 1, (oldValue, newValue) -> oldValue + newValue);

servedClients++;

return true;

}

public synchronized void returnCar() {

availableCars++;

}

public Map<Integer, Integer> getStats() {

return stats;

}

public int getServedClients() {

return servedClients;

}

}

Файл Client.java :

package org.example;

import java.util.Random;

public class Client implements Runnable {

private static final Random R = new Random();

private static final int MIN = 4, MAX = 48, STEP = 2;

private final CarSharing service;

private final int id;

public Client(CarSharing service, int id) {

this.service = service;

this.id = id;

}

@Override

public void run() {

int duration = MIN + STEP \* R.nextInt((MAX - MIN) / STEP + 1);

if (service.tryRentCar(duration)) {

System.out.printf("Клиент %02d взял машину на %2d ч%n", id, duration);

try {

Thread.sleep(duration \* 100);

} catch (InterruptedException ignored) {}

service.returnCar();

} else {

System.out.printf("Клиент %02d ушёл — машин нет%n", id);

}

}

}

Файл Simulation.java :

package org.example;

import java.util.HashMap;

import java.util.Map;

public class Simulation {

public static void main(String[] args) throws InterruptedException {

int totalCars = 30;

CarSharing service = new CarSharing(totalCars);

Thread[] clients = new Thread[24];

for (int i = 0; i < 24; i++) {

Thread.sleep(100); // 1 час = 100 мс

clients[i] = new Thread(new Client(service, i + 1));

clients[i].start();

}

for (Thread t : clients) {

t.join();

}

System.out.println("\nКлиентов обслужено: " + service.getServedClients());

System.out.println("Статистика по длительности (часов → число):");

for (Map.Entry<Integer, Integer> e : service.getStats().entrySet().stream()

.sorted(Map.Entry.comparingByKey())

.toList()) {

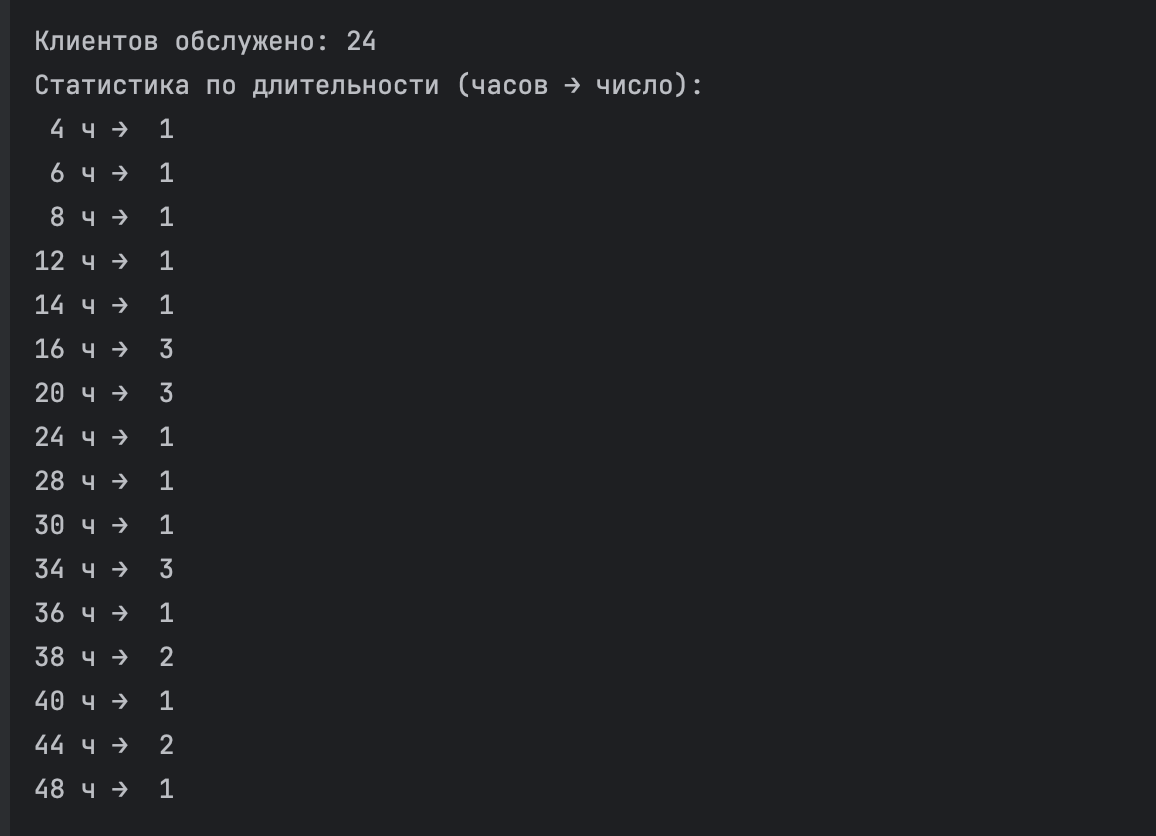
System.out.printf("%2d ч → %2d%n", e.getKey(), e.getValue());

}

}

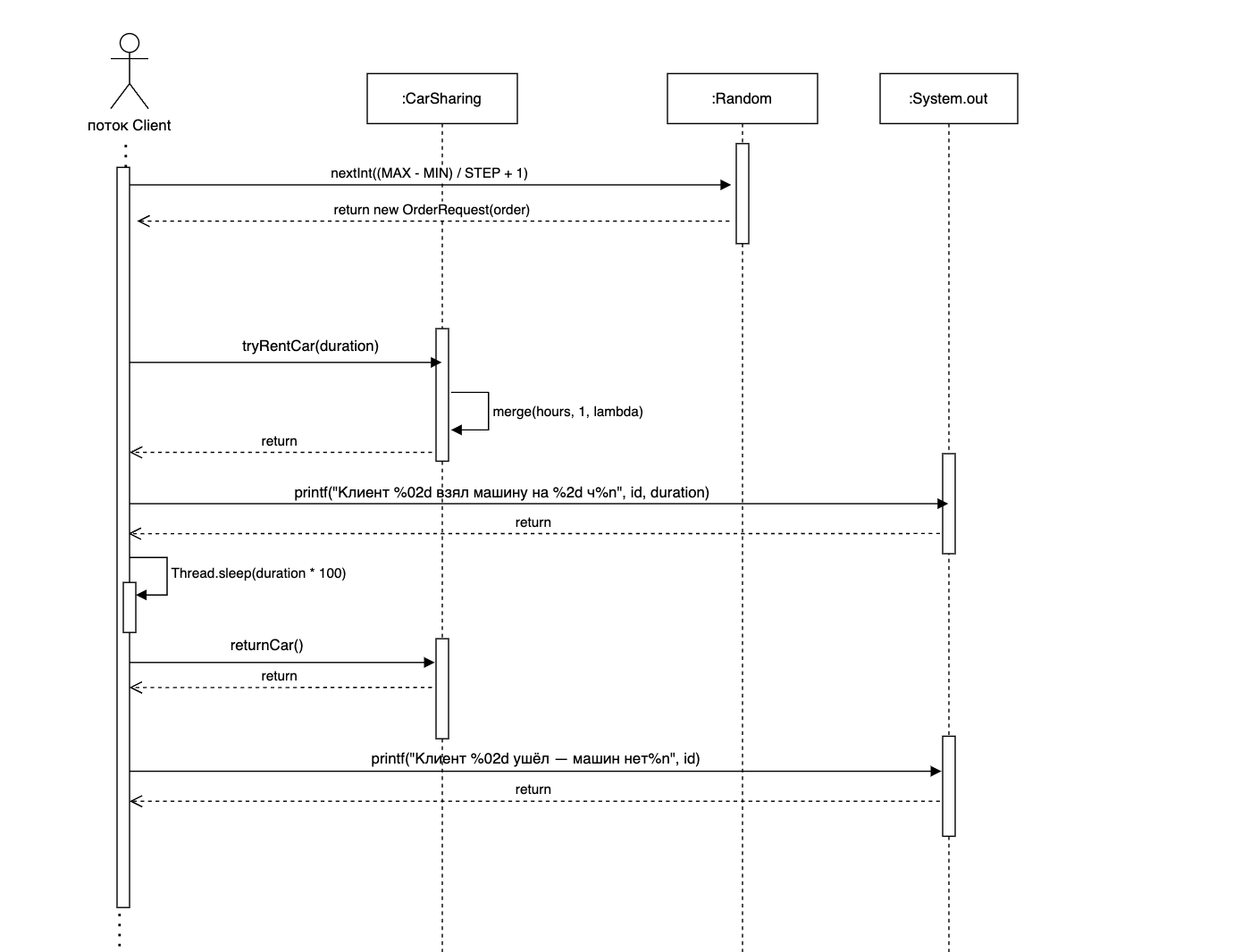
}

1. **Результат выполнения программы**



1. **Диаграмма последовательности**

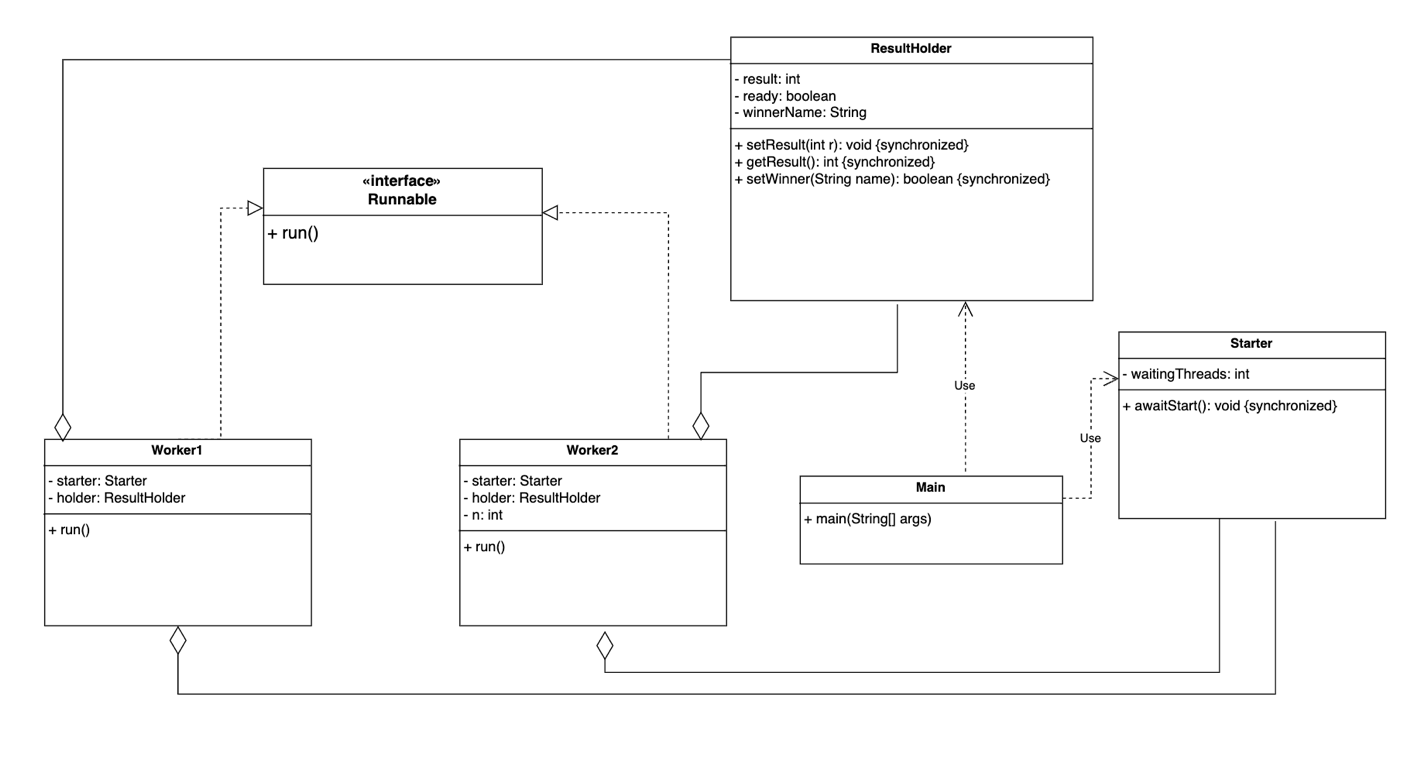
Действия, выполняемые в процессе выполнения каждого из потоков можно представить в виде диаграммы последовательности:



**Задача :**

Создать главный поток и два рабочих, где: главный ждет первого рабочего с join(), принимает число (некоторый результат расчета) от первого и передает его второму. Далее второй и первый выполняют на «кто быстрее» одинаковые задания с этим числом, причем Второй рабочий использует yield(), а Первый рабочий использует sleep().

1. **Диаграмма классов**

****

1. **Листинг программы**

Файл Worker1:

package task2;

public class Worker1 implements Runnable{

private final Starter starter;

private final ResultHolder holder;

public Worker1(Starter s, ResultHolder h) {

this.starter = s;

this.holder = h;

}

@Override

public void run() {

String name = Thread.currentThread().getName();

int result = 0;

for (int i = 1; i <= 100; i++) result += i;

System.out.println(name + ": вычислил " + result);

holder.setResult(result); // Сигнал main, что число готово

// --- Фаза 2: гонка ---

try {

starter.awaitStart();

} catch (InterruptedException e) {

return;

}

long sum = 0;

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

sum += i;

try { Thread.sleep(1); } catch (InterruptedException ignored) {}

}

if (holder.setWinner(name)) {

System.out.println(name + " - (победитель)"+ " завершил sum=" + sum);

} else {

System.out.println(name + " завершил sum=" + sum);

}

}

}

Файл Worker2:

package task2;

public class Worker2 implements Runnable {

private final Starter starter;

private final ResultHolder holder;

private final int n;

public Worker2(Starter s, ResultHolder h,int n) {

this.starter = s;

this.holder = h;

this.n= n;

}

@Override

public void run() {

String name = Thread.currentThread().getName();

try {

starter.awaitStart();

} catch (InterruptedException e) {

return;

}

long sum = 0;

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

sum += i;

Thread.yield();

}

if (holder.setWinner(name)) {

System.out.println(name+ " - (победитель)"+ " завершил sum=" + sum);

} else {

System.out.println(name + " завершил sum=" + sum);

}

}

}

Файл ResultHolder:

package task2;

public class ResultHolder {

private int result;

private boolean ready = false;

private String winnerName;

public synchronized void setResult(int r) {

result = r;

ready = true;

notifyAll();

}

public synchronized int getResult() throws InterruptedException {

while (!ready) {

wait();

}

return result;

}

public synchronized boolean setWinner(String name) {

if (winnerName == null) {

winnerName = name;

return true;

}

return false;

}

}

Файл Starter:

package task2;

public class Starter {

private int waitingThreads = 0;

public synchronized void awaitStart() throws InterruptedException {

waitingThreads++;

if (waitingThreads < 2) {

wait(); // Первый поток ждёт

} else {

notifyAll(); // Второй пришёл — будим всех

}

}

}

Файл Main:

package task2;

public class Main {

public static void main(String[] args) throws InterruptedException {

Starter starter = new Starter();

ResultHolder holder = new ResultHolder();

Worker1 w1 = new Worker1(starter, holder);

Thread t1 = new Thread(w1);

t1.setName("Worker1");

t1.start();

int result = holder.getResult();

System.out.println("Main: получил от Worker1 число = " + result);

Worker2 w2 = new Worker2(starter, holder, result);

Thread t2 = new Thread(w2);

t2.setName("Worker2");

t2.start();

System.out.println("Main: началась гонка");

t1.join();

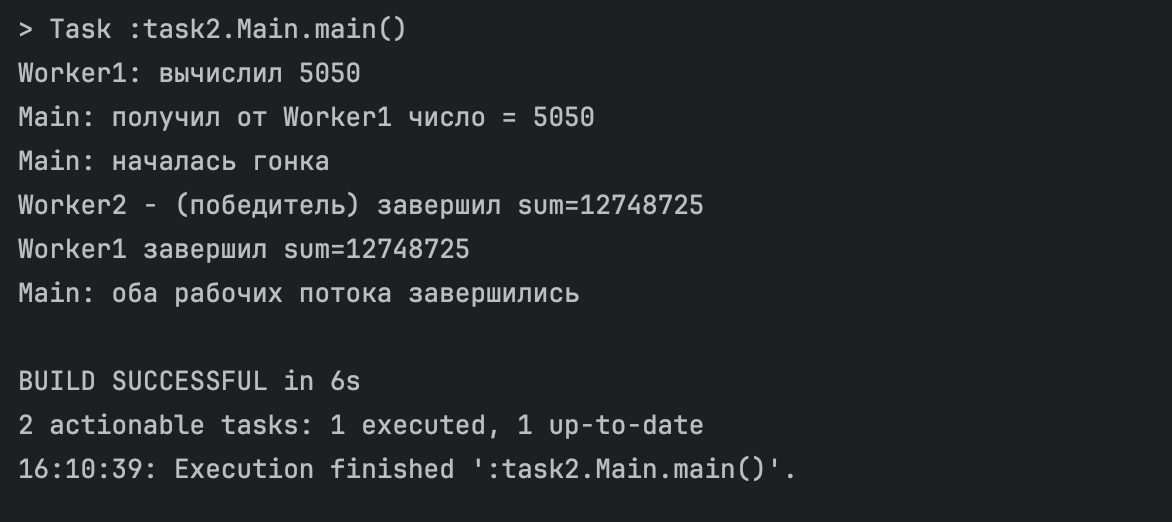
t2.join();

System.out.println("Main: оба рабочих потока завершились");

}

}

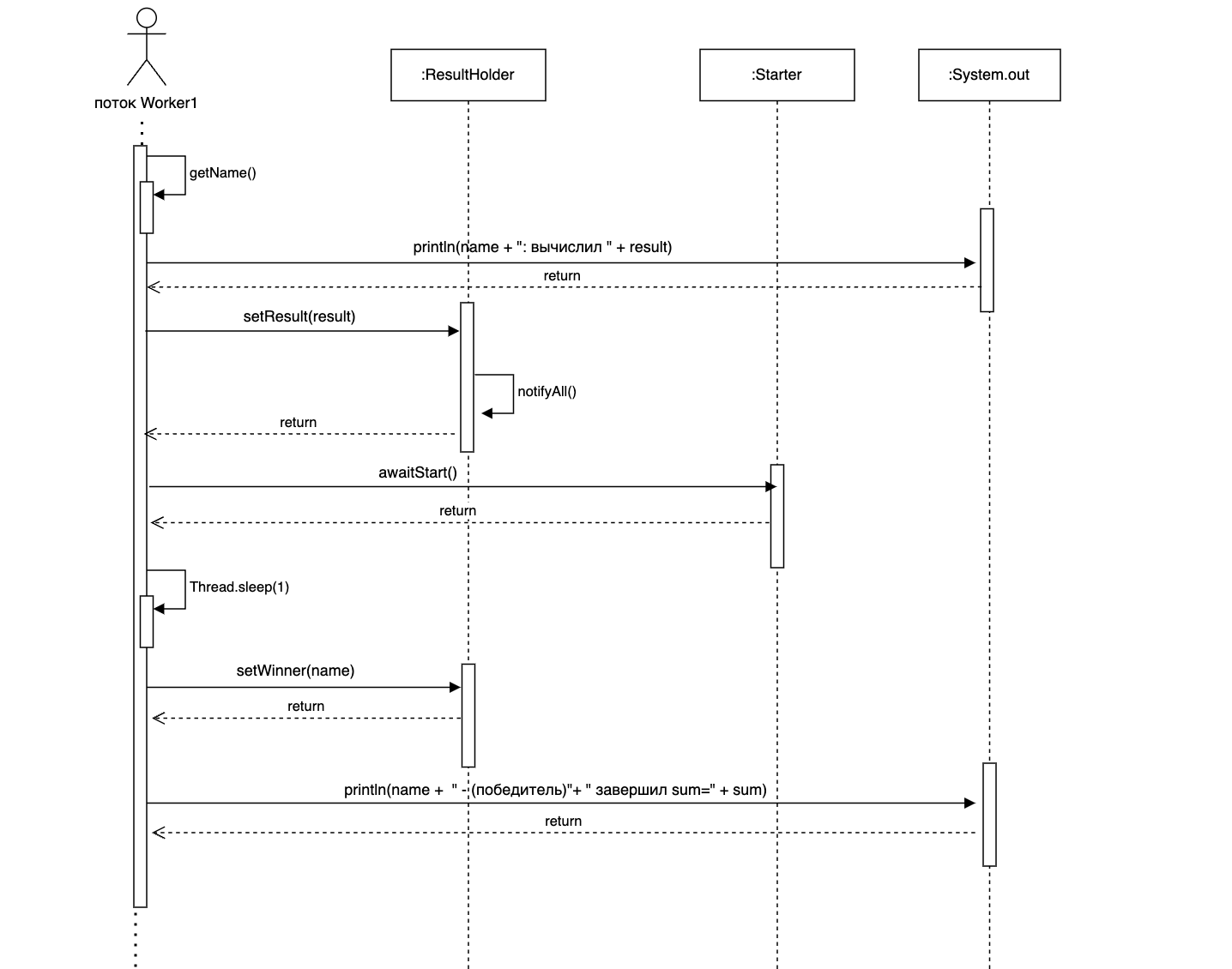
1. **Результат выполнения программы**

****

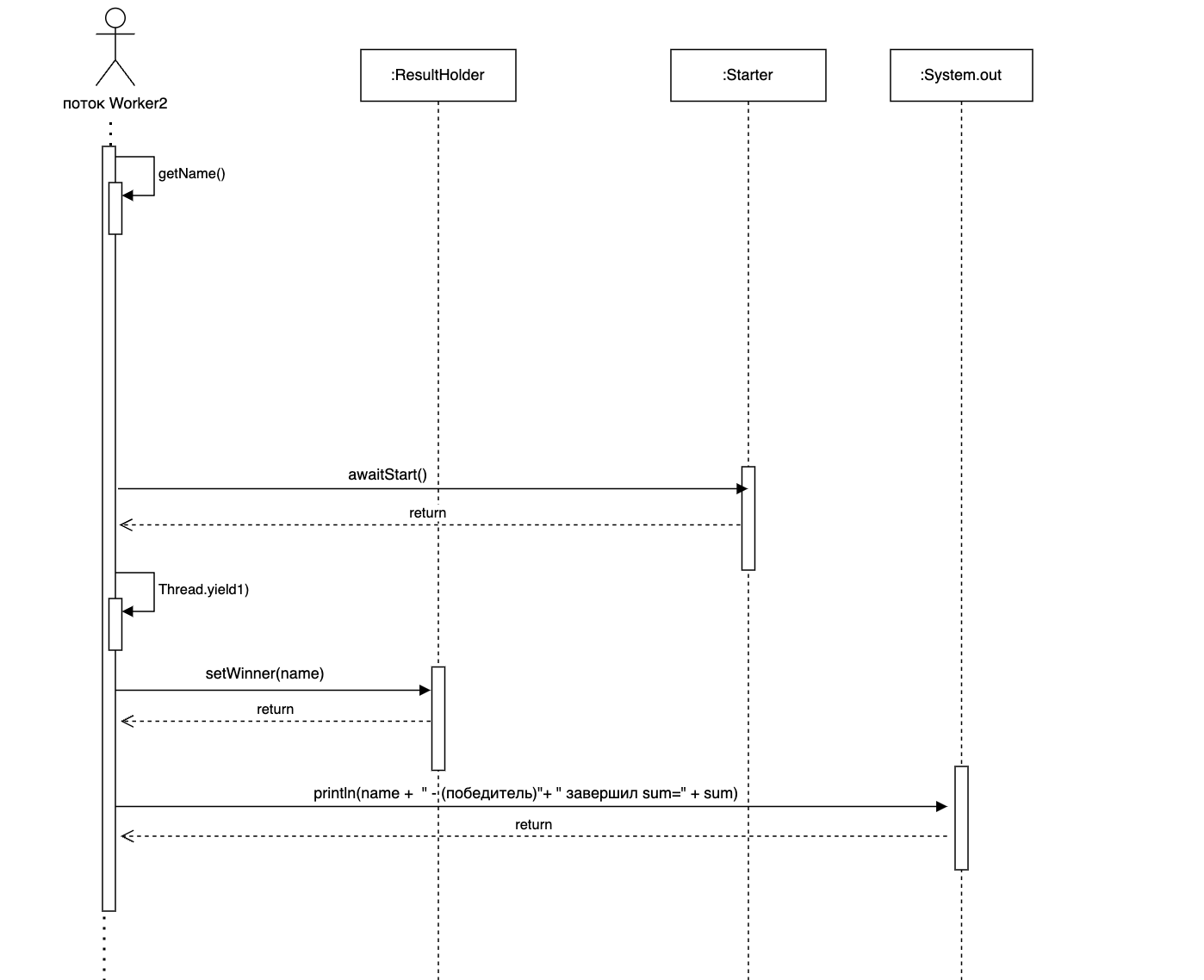
1. **Диаграмма последовательности**

Действия, выполняемые в процессе выполнения каждого из потоков можно представить в виде диаграммы последовательности:

Для рабочего потока 1:



Для рабочего потока 2:

****

**Вывод:**

В ходе лабораторной работы мы исследовали приемы и методы организации и управления потоками.