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**

Работу выполнил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чеуж А.А.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

Краснодар

2025

**Тема**: Потоки.

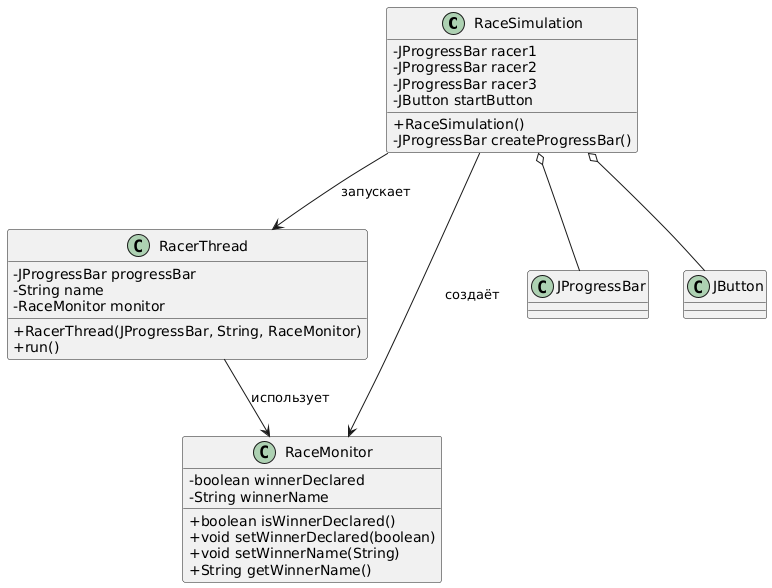
**Цель работы**: Изучить способы организации многопоточной работы приложений на Java.

**Ход работы:**

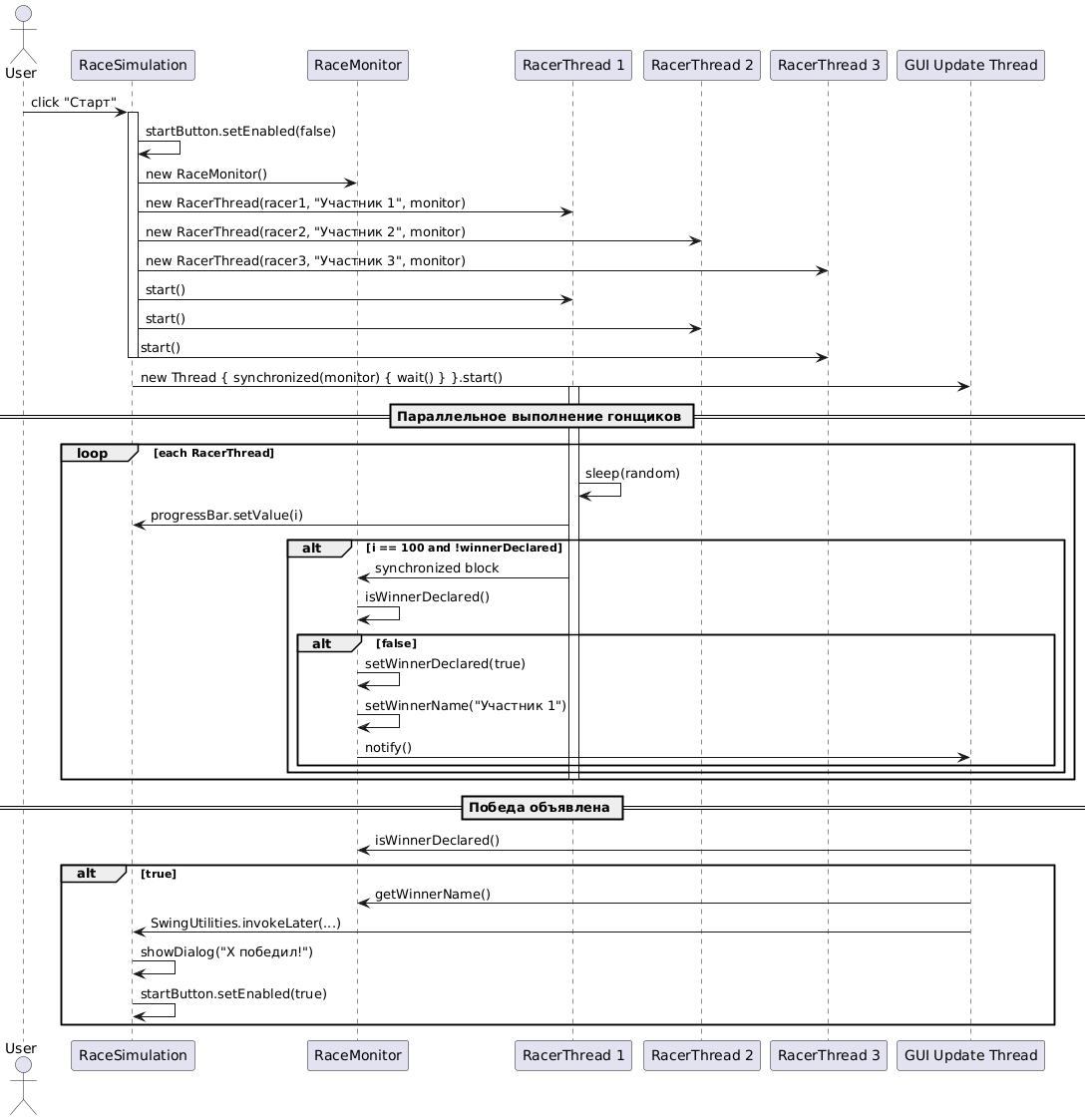
Задание:

**Гонки**. 3 участника наперегонки наращивают каждый свой ProgressBar на разное случайное число единиц прогресса. По окончании должно появиться сообщение о том, кто прибыл первым, кто последним. Вариант графический.

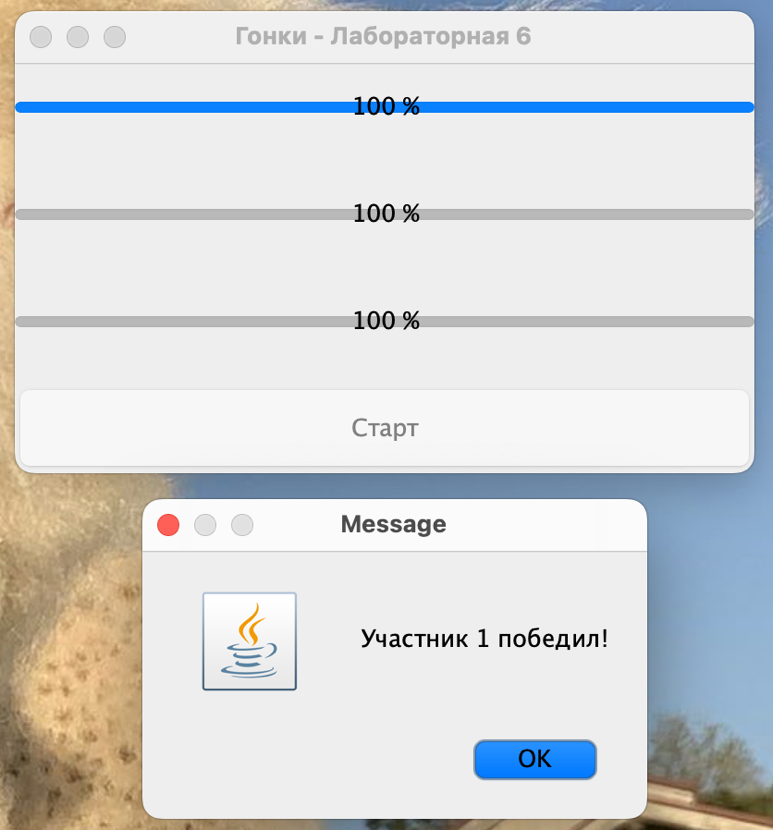
Была построена UML-диаграмма классов:



Для упрощения понимания работы программы была построена диаграмма потоков:



Пример работы программы:



Листнинг программы:

import java.awt.\*;

import javax.swing.\*;

public class RaceSimulation extends JFrame {

private JProgressBar racer1, racer2, racer3;

private JButton startButton;

public RaceSimulation() {

setTitle("Гонки - Лабораторная 6");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(400, 250);

setLocationRelativeTo(null);

racer1 = createProgressBar();

racer2 = createProgressBar();

racer3 = createProgressBar();

startButton = new JButton("Старт");

startButton.addActionListener(e -> {

startButton.setEnabled(false);

RaceMonitor monitor = new RaceMonitor();

RacerThread t1 = new RacerThread(racer1, "Участник 1", monitor);

RacerThread t2 = new RacerThread(racer2, "Участник 2", monitor);

RacerThread t3 = new RacerThread(racer3, "Участник 3", monitor);

t1.start();

t2.start();

t3.start();

// Создаем управляющий поток

new Thread(() -> {

synchronized (monitor) { // захватывает объект monitor

try {

while (!monitor.isWinnerDeclared()) {

monitor.wait(); // освобождает monitor и ждёт, пока поток гонщика вызовет notify()

}

} catch (InterruptedException ex) {

throw new RuntimeException(ex);

}

}

// Победитель найден, обновляем GUI

SwingUtilities.invokeLater(() -> {

JOptionPane.showMessageDialog(null, monitor.getWinnerName() + " победил!");

startButton.setEnabled(true);

});

}).start();

});

setLayout(new GridLayout(4, 1, 10, 10));

add(racer1);

add(racer2);

add(racer3);

add(startButton);

setVisible(true);

}

private JProgressBar createProgressBar() {

JProgressBar bar = new JProgressBar(0, 100);

bar.setStringPainted(true);

bar.setPreferredSize(new Dimension(350, 30));

return bar;

}

public static void main(String[] args) {

SwingUtilities.invokeLater(RaceSimulation::new);

}

}

import javax.swing.\*;

public class RacerThread extends Thread {

private final JProgressBar progressBar;

private final String name;

private final RaceMonitor monitor;

public RacerThread(JProgressBar bar, String name, RaceMonitor monitor) {

this.progressBar = bar;

this.name = name;

this.monitor = monitor;

}

@Override

public void run() {

System.out.println(name + " стартовал!");

for (int i = 0; i <= 100; i++) {

try {

Thread.sleep((long) (Math.random() \* 100)); // рандомная скорость гонщинка

} catch (InterruptedException e) {

return;

}

progressBar.setValue(i);

if (i == 100) {

synchronized (monitor) { // в случае победы поток захватывает объект

if (!monitor.isWinnerDeclared()) {

monitor.setWinnerDeclared(true);

System.out.println(name + " победил!");

monitor.setWinnerName(name);

monitor.notify(); // будим управляющий поток

}

}

}

}

System.out.println(name + " закончил гонку.");

}

}

public class RaceMonitor {

private boolean winnerDeclared = false;

private String winnerName = null;

public synchronized boolean isWinnerDeclared() {

return winnerDeclared;

}

public synchronized void setWinnerDeclared(boolean value) {

this.winnerDeclared = value;

}

public synchronized void setWinnerName(String name) {

this.winnerName = name;

}

public synchronized String getWinnerName() {

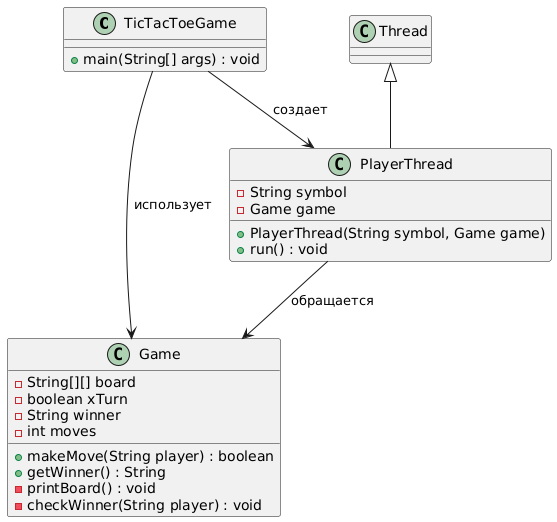
return winnerName;

}

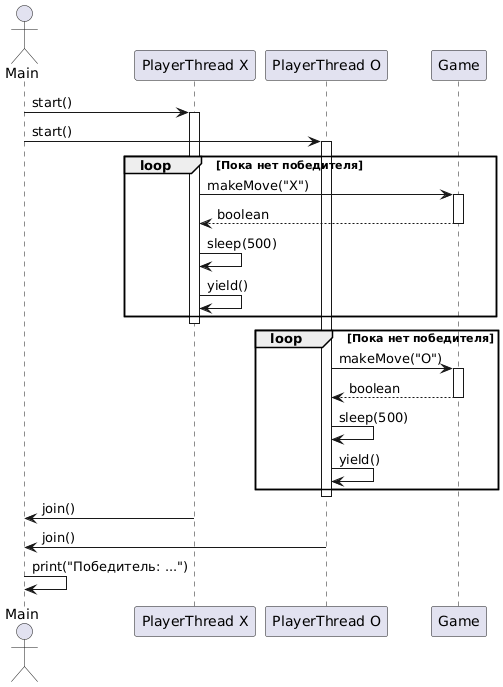
}

**Крестики – нолики**. Упрощенный вариант – консольный. Должны выдаваться сообщения: 1-й поток окончил! 2-й поток окончил! И имя выигравшего потока, если была выигрышная комбинация.

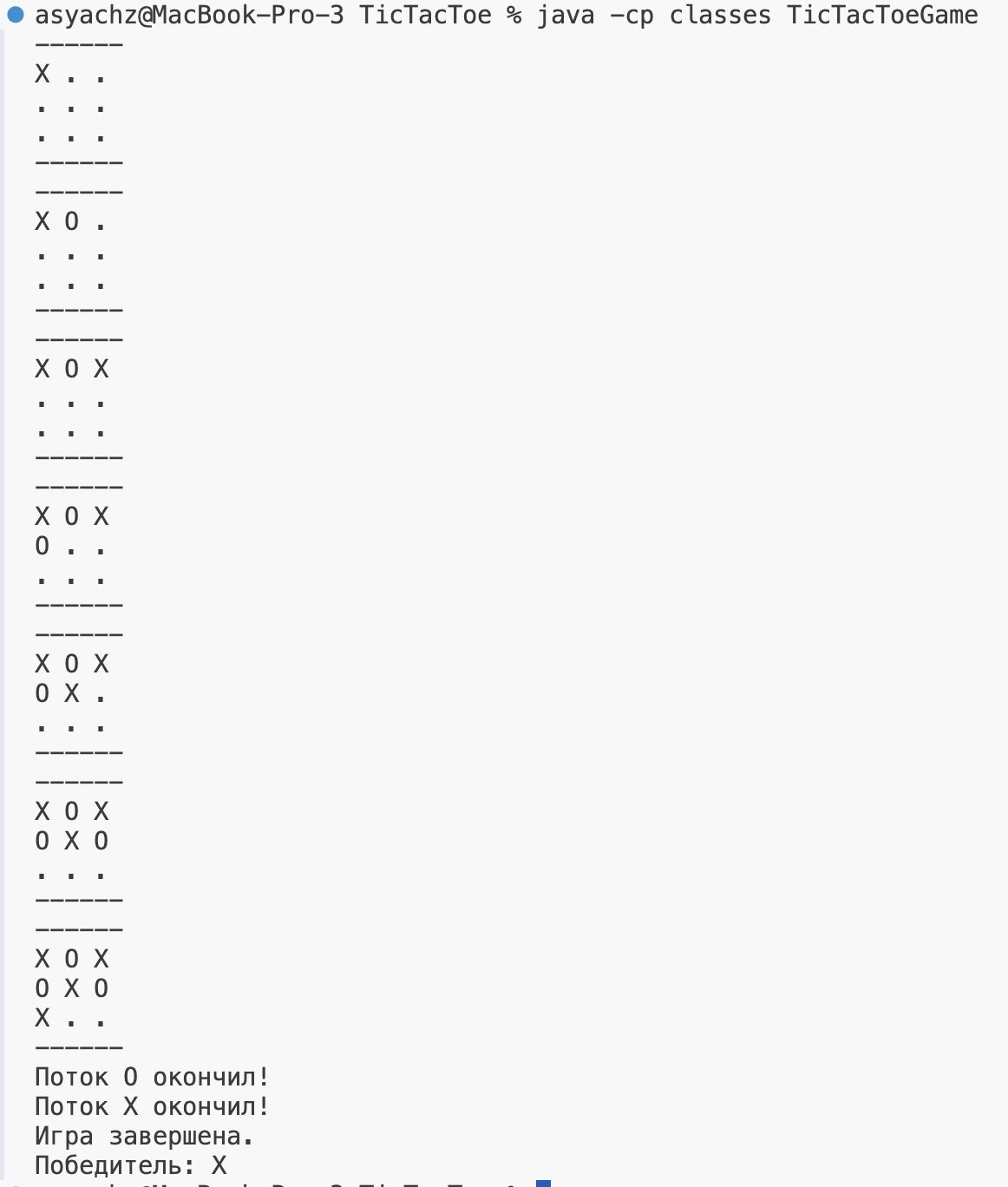
Была построена UML-диаграмма классов:



Для упрощения понимания работы программы была построена диаграмма потоков:



Пример работы программы:



Листнинг программы:

public class TicTacToeGame {

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

Game game = new Game();

Thread player1 = new PlayerThread("X", game);

Thread player2 = new PlayerThread("O", game);

player1.start();

player2.start();

// Главный поток ждет завершения player1 и player2

// Метод join() приостанавливает выполнение текущего потока

player1.join();

player2.join();

System.out.println("Игра завершена.");

System.out.println("Победитель: " + game.getWinner());

}

}

class PlayerThread extends Thread {

private final String symbol;

private final Game game;

public PlayerThread(String symbol, Game game) {

this.symbol = symbol;

this.game = game;

}

@Override

public void run() {

while (game.getWinner().equals("Ничья")) {

boolean moved = game.makeMove(symbol);

if (!moved) break;

try {

Thread.sleep(500);

Thread.yield(); // передаем управление другому потоку

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Поток " + symbol + " окончил!");

}

}

public class Game {

private final String[][] board = new String[3][3];

private boolean xTurn = true; // true — X, false — O

private String winner = null;

private int moves = 0;

// Модификатор synchronized гарантирует, что только один поток может

// одновременно выполнять этот метод на одном и том же объекте

public synchronized boolean makeMove(String player) {

while ((xTurn && player.equals("O")) || (!xTurn && player.equals("X"))) {

try {

wait(); // Ждём своей очереди

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if (winner != null || moves >= 9) return false;

// Найдём первую свободную клетку

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == null) {

board[i][j] = player;

moves++;

printBoard();

checkWinner(player);

xTurn = !xTurn;

notifyAll(); // Разрешаем следующему

return true;

}

}

}

return false;

}

private void printBoard() {

System.out.println("------");

for (String[] row : board) {

for (String cell : row) {

System.out.print((cell != null ? cell : ".") + " ");

}

System.out.println();

}

System.out.println("------");

}

private void checkWinner(String player) {

for (int i = 0; i < 3; i++) {

if (

player.equals(board[i][0]) && player.equals(board[i][1]) && player.equals(board[i][2]) ||

player.equals(board[0][i]) && player.equals(board[1][i]) && player.equals(board[2][i])

) {

winner = player;

}

}

if (

player.equals(board[0][0]) && player.equals(board[1][1]) && player.equals(board[2][2]) ||

player.equals(board[0][2]) && player.equals(board[1][1]) && player.equals(board[2][0])

) {

winner = player;

}

}

public synchronized String getWinner() {

return winner != null ? winner : "Ничья";

}

}

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