**Лабораторная работа №6: Потоки**

**Вавакин Владислав**

**Постановка задачи.**

**1) Спорщики.** Три участника путешествия наперебой выбирают маршрут (суша, море, воздух). По окончании объявляется победитель и выбранный способ перемещения.

**2) Слова.** Пять потоков в строгой очередности считывают слова из одного и того же файла (Поток1: слово1, слово5, слово9; Поток2: слово2, слово6, слово10 и т.д. )в свой массив слов, затем выводят в консоль почитанное: сначала все слова потока1, затем потока 2, и т.д.

**Ход работы**

Задача 1:

Необходимо смоделировать процесс гонки данных, где каждый из трёх участников выбирает со случайной задержкой один из трёх путей передвижения, и побеждает тот, кто выберет первым.

Напишем класс Race, содержащий основные методы гонки:

- setWinner – устанавливает имя победителя и выбранный путь, если они ещё не заданы;

- addChoice – добавляет в хэш имя участника и его выбранный путь передвижения;

- finishParticipant – увеличивает счётчик завершивших работу процессов-участников;

- getFinishedCount – возвращает количество завершивших работу процессов;

- getWinnerName, getWinnerTransport – возвращает имя и путь передвижения победителя;

- getAllChoices – возвращает всех участников и выбранные пути передвижения.

import java.util.HashMap;

import java.util.Map;

class Race {

private String winnerName;

private String winnerTransport;

private int finishedCount = 0;

private final Map<String, String> choices = new HashMap<>();

public synchronized boolean setWinner(String name, String transport) {

if (winnerName == null) {

winnerName = name;

winnerTransport = transport;

return true;

}

return false;

}

public synchronized void addChoice(String name, String transport) {

choices.put(name, transport);

}

public synchronized void finishParticipant() {

finishedCount++;

}

public synchronized int getFinishedCount() {

return finishedCount;

}

public String getWinnerName() {

return winnerName;

}

public String getWinnerTransport() {

return winnerTransport;

}

public Map<String, String> getAllChoices() {

return new HashMap<>(choices);

}

}

Напишем класс Participant, наследующий интерфейс Runnable и реализующий метод run, в котором происходит случайная задержка, после которой выбирается путь передвижения, затем выбранный путь вносится в объект класса гонки, после чего в блоке с synchronized выводится информация о потоке и вызывается finishParticipant и notifyAll из объекта race.

import java.util.Random;

class Participant implements Runnable {

private static final String[] TRANSPORTS = {"суша", "море", "воздух"};

private final String name;

private final Race race;

private final Random random = new Random();

public Participant(String name, Race race) {

this.name = name;

this.race = race;

}

@Override

public void run() {

try {

Thread.sleep(random.nextInt(500));

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

String transport = TRANSPORTS[random.nextInt(TRANSPORTS.length)];

race.addChoice(name, transport);

synchronized (race) {

if (race.setWinner(name, transport)) {

System.out.println(name + " выбрал " + transport + " (победитель)");

} else {

System.out.println(name + " выбрал " + transport);

}

race.finishParticipant();

race.notifyAll();

}

}

}

Главный класс программы – TravelRace, создающий три потока participants и гонку Race. Каждый из потоков запускается, после чего до тех пор, пока все 3 участника не завершат работу, в race вызывается метод wait. По завершении работы всех потоков в консоль выводится информация о всех участниках и их выбранных путях передвижения.

public class TravelRace {

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

Race race = new Race();

Thread[] participants = {

new Thread(new Participant("Участник 1", race)),

new Thread(new Participant("Участник 2", race)),

new Thread(new Participant("Участник 3", race))

};

for (Thread t : participants) t.start();

synchronized (race) {

while (race.getFinishedCount() < 3) {

race.wait();

}

}

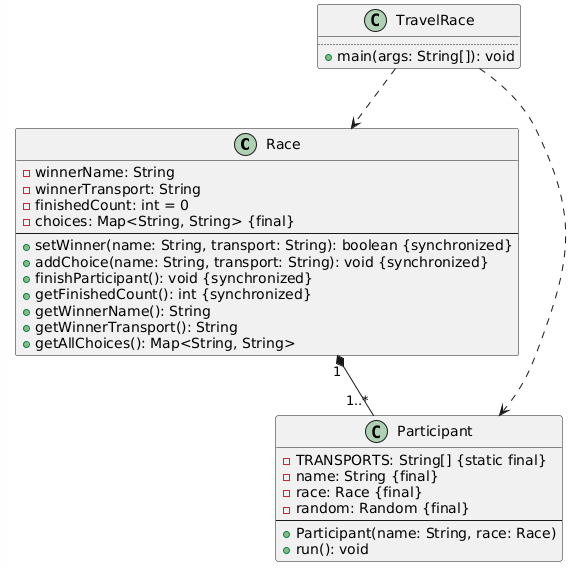
System.out.println("\nПобедитель: " + race.getWinnerName()

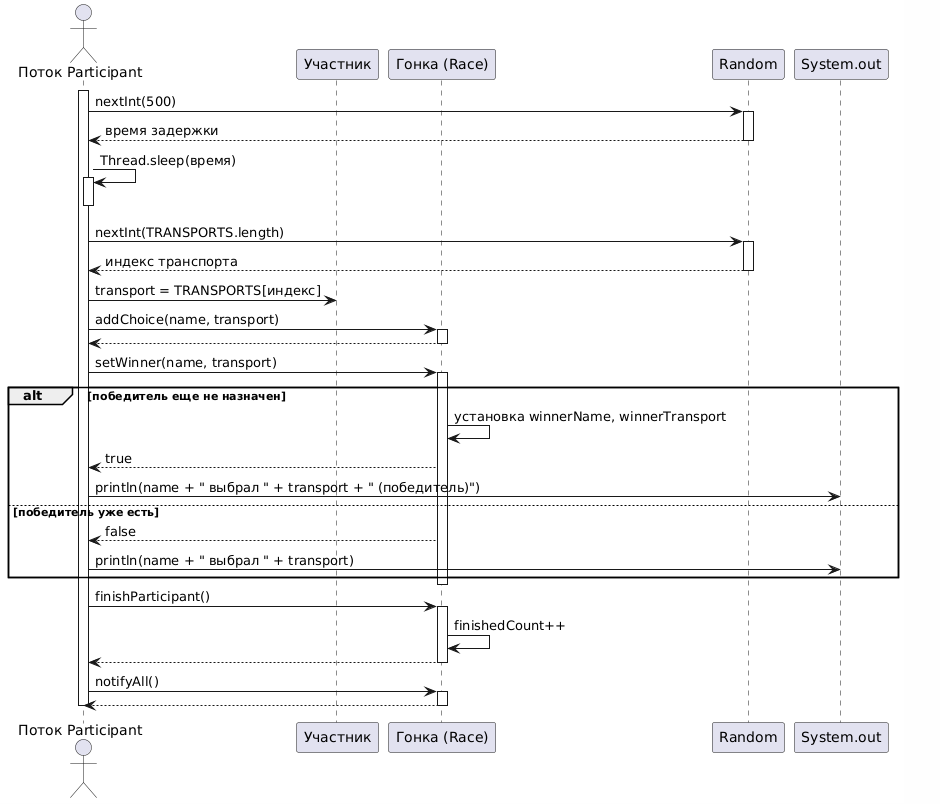
+ ", способ: " + race.getWinnerTransport());

System.out.println("Все выборы: " + race.getAllChoices());

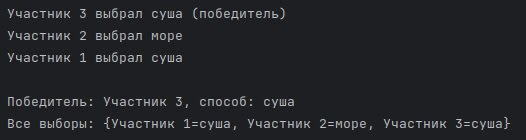
}

}

Диаграмма классов имеет вид:  


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


Результат выполнения программы:



Задача 2: необходимо, чтобы каждый из потоков поочерёдно считывал по слову из исходного файла. Для этого напишем класс FileController, хранящий количество запущенных потоков, список слов каждого из потоков, номер текущего потока, объект BufferedReader (содержит метод ReadLine) и флаг, обозначающий достиг ли процесс конца файла.

Потоки, читающие файл, вызывают метод readWords с собственным threadId. Если этот threadId совпадает с текущим номером в очереди, то считывается строка из файла и вносится в соответствующий потоку список, иначе поток ждёт команды notify.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

class FileController {

private final int threadCount;

private final List<List<String>> threadWords;

private int currentThread = 0;

private BufferedReader reader;

private boolean fileFinished = false;

public FileController(int threadCount) {

this.threadCount = threadCount;

this.threadWords = new ArrayList<>();

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

threadWords.add(new ArrayList<>());

}

try {

this.reader = new BufferedReader(new FileReader(SequentialFileReaders.FILE\_PATH));

} catch (IOException e) {

System.err.println("Error opening file: " + e.getMessage());

System.exit(1);

}

}

public synchronized void readWords(int threadId) throws IOException, InterruptedException {

while (!fileFinished) {

while (currentThread != threadId && !fileFinished) {

wait();

}

if (fileFinished) {

break;

}

String line = reader.readLine();

if (line == null) {

fileFinished = true;

notifyAll();

break;

}

threadWords.get(threadId).add(line.trim());

currentThread = (currentThread + 1) % threadCount;

notifyAll();

}

}

public void printResults() {

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

System.out.println("Thread " + (i + 1) + " words:");

for (String word : threadWords.get(i)) {

System.out.println(word);

}

System.out.println();

}

}

}

Далее напишем главный класс, объявляющий потоки в виде объектов класса Thread с передаваемой лямбда-функцией readWords. Далее он запускает все потоки и с помощью join ожидает их завершения, после чего выводит результаты на экран.

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class SequentialFileReaders {

private static final int THREAD\_COUNT = 5;

public static final String FILE\_PATH = "./Lab6\_2/words.txt";

public static void main(String[] args) {

FileController controller = new FileController(THREAD\_COUNT);

List<Thread> threads = new ArrayList<>();

for (int i = 0; i < THREAD\_COUNT; i++) {

final int threadId = i;

Thread thread = new Thread(() -> {

try {

controller.readWords(threadId);

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

});

threads.add(thread);

thread.start();

}

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

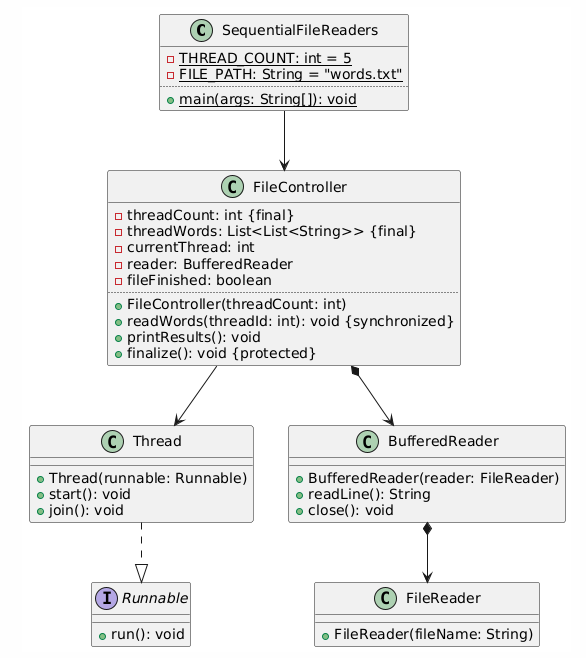
}

controller.printResults();

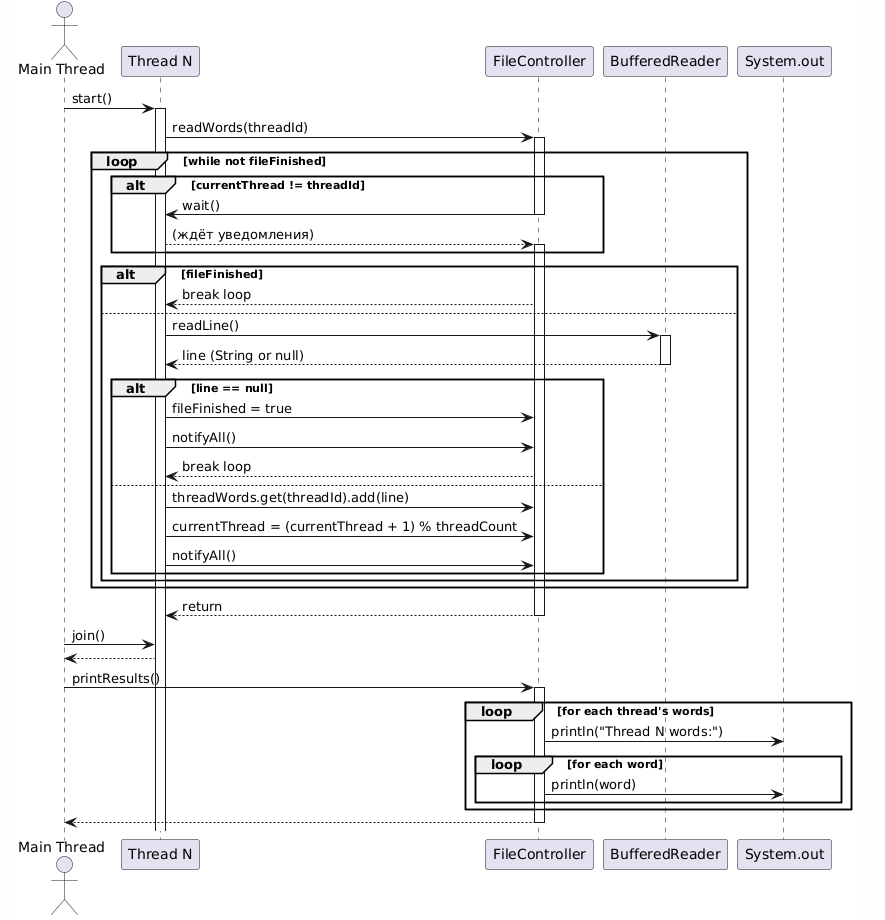
}

}

Диаграмма классов программы имеет вид:



Процесс выполнения одного потока выражается следующей диаграммой последовательности:



Результат выполнения программы:

