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 №5**

**Дисциплина: Программирование для мобильных платформ**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.А. Приходько

**Тема работы:** Многопоточность.

**Задача:** Написать игру 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 выводят в консоль сначала каждый свой массив, потом весь считанный текст в обратном порядке: слово20, слово19, слово18, слово17…

**Описание решения:**

Игра реализована с использованием 5 потоков (TOTAL\_THREADS), каждый из которых обрабатывает 4 слова (WORDS\_PER\_THREAD). Для синхронизации доступа к общим ресурсам между потоками используется замок (ReentrantLock). Для хранения слов создан массив wordArrays, представляющий списки слов для каждого потока, а также общий список allWords. Для чтения слов из файла применяется BufferedReader, где каждый поток считывает слова из файла в бесконечном цикле.

Для обеспечения синхронизированного доступа к файлу используется синхронизированный блок, гарантирующий, что только один поток может читать слово из файла в определенный момент. Переключение между потоками осуществляется с использованием переменной currentPlayer, представляющей идентификатор текущего активного потока. Потоки ожидают своей очереди, читают слово, добавляют его в свой список, затем уведомляют другие потоки о завершении операции.

Для ожидания завершения работы всех потоков в основном методе используется метод join(). После завершения всех потоков выводится список слов для каждого потока, а также формируется и выводится список всех слов в обратном порядке.

Обработка исключений включает в себя обработку IOException и InterruptedException с выводом трассировки стека.

На рисунке 1 представлена UML-диаграмма классов.

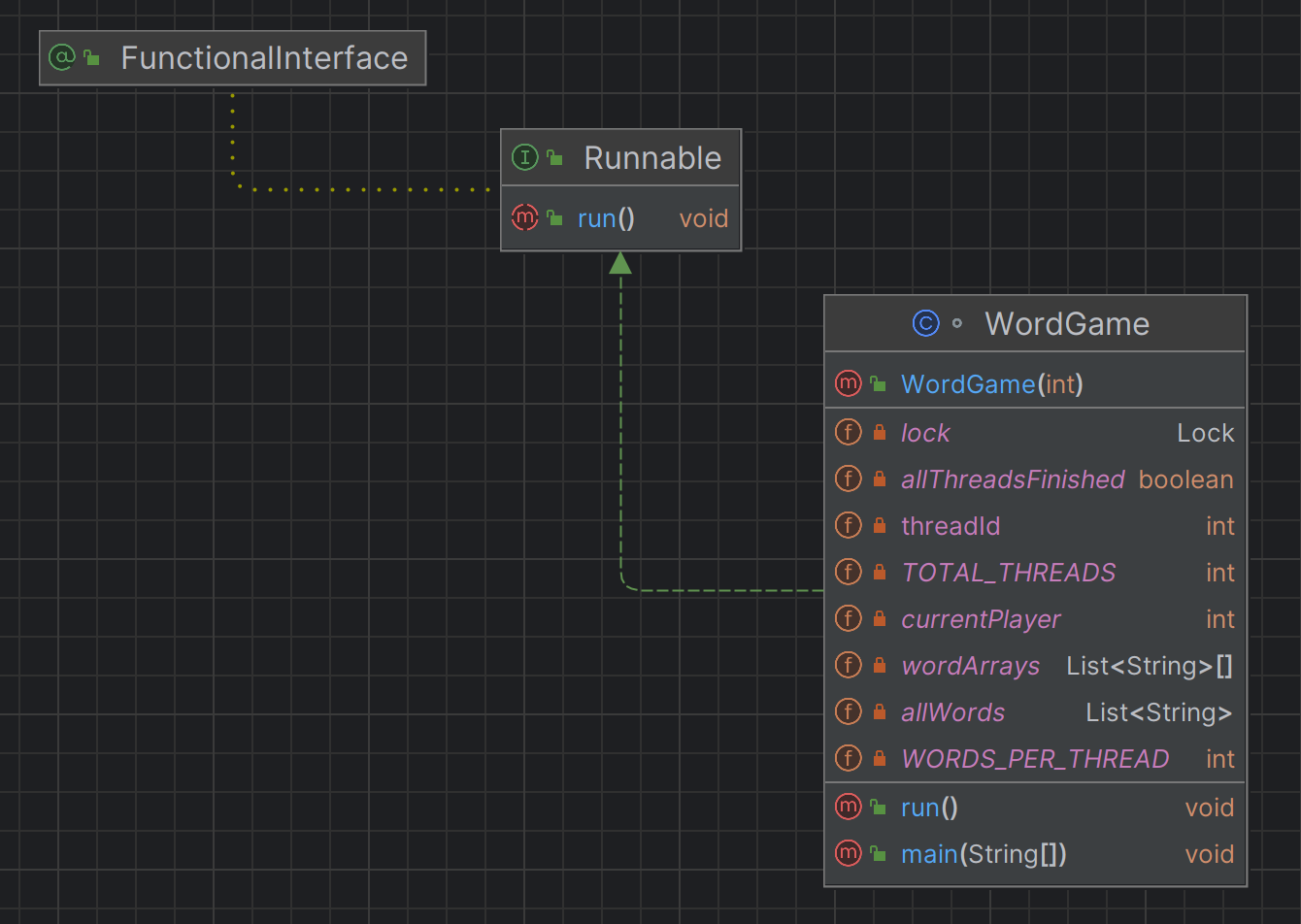


Рисунок 1 – UML-диаграмма

На диаграмме видно связь Runnable и WordGame - класс WordGame реализует интерфейс Runnable, что позволяет объектам этого класса быть исполнимыми в отдельных потоках. Интерфейс Runnable имеет единственный метод run(), который содержит код, который будет выполнен при запуске потока.

**Работа потоков:**

В классе WordGame создается общий замок lock типа ReentrantLock, который будет использоваться для синхронизации доступа к общим ресурсам.

Для идентификации потоков используем:

currentPlayer - идентификатор текущего активного потока.

allThreadsFinished - флаг, сигнализирующий об окончании работы всех потоков.

**Метод run()** представляет из себя бесконечный цикл для чтения слов из файла, синхронизированный блок с использованием замка для синхронизации потоков.

Поток ожидает своей очереди или окончания работы всех потоков,читает слово из файла, добавляет его в список текущего потока и выводит в консоль, затем переключает активный поток и уведомляет другие потоки.Выход из цикла при окончании слов в файле или завершении всех потоков.

**Метод main()** создает и запускает массив потоков threads, каждый из которых выполняет экземпляр WordGame,ожидает завершения работы всех потоков,выводит слова для каждого потока,формирует и выводит список всех слов в обратном порядке.

**Особенности работы потоков:**

* Используется synchronized блок для обеспечения безопасного доступа к общим данным между потоками.
* Потоки считывают слова из файла в соответствии с их очередностью (currentPlayer), что обеспечивает равномерное распределение слов между потоками.
* Потоки ожидают своей очереди, когда другой поток активен. Уведомление происходит после того, как текущий поток обработал
* Метод join() используется для ожидания завершения работы каждого потока перед продолжением выполнения.

На рисунке 2 видно, как выглядит файл со словами изначально.

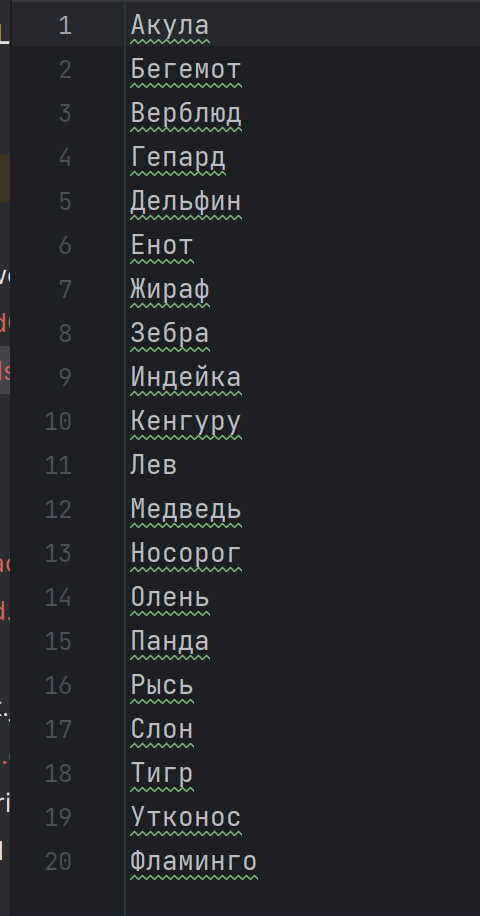


Рисунок 2 – Файл со словами.

На рисунке 3 показана работа с потоками, каждый поток считывает по одному слову по очереди.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 3 – Считывание слов потоками.

На рисунке 4 показан вывод списка слов для каждого потока.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3 – Массивы слов для каждого потока.

На рисунке 4 показан вывод общего списка слов в обратном порядке.

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 4 – Вывод списка слов в обратном порядке

.

**Листинг программ**

Файл WordGame.java

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
// Класс WordGame реализует интерфейс Runnable и представляет игру со словами, выполняемую несколькими потоками.  
class WordGame implements Runnable {  
 private static final int *TOTAL\_THREADS* = 5;  
 private static final int *WORDS\_PER\_THREAD* = 4;  
  
 // Общий замок для синхронизации потоков.  
 private static final Lock *lock* = new ReentrantLock();  
  
 // Массив списков слов для каждого потока.  
 private static List<String>[] *wordArrays* = new List[*TOTAL\_THREADS*];  
  
 // Общий список для хранения всех слов.  
 private static List<String> *allWords* = new ArrayList<>();  
  
 // Идентификатор текущего активного потока.  
 private static int *currentPlayer* = 1;  
  
 // Флаг, сигнализирующий об окончании работы всех потоков.  
 private static boolean *allThreadsFinished* = false;  
  
 // Идентификатор текущего потока.  
 private int threadId;  
  
 // Конструктор принимает и устанавливает идентификатор текущего потока.  
 public WordGame(int threadId) {  
 this.threadId = threadId;  
 // Инициализация списка слов для текущего потока.  
 *wordArrays*[threadId - 1] = new ArrayList<>();  
 }  
  
 // Метод, который выполняется при запуске потока.  
 @Override  
 public void run() {  
 try (BufferedReader reader = new BufferedReader(new FileReader("src/Game\_words/words.txt"))) {  
 int wordCount = 0;  
 // Бесконечный цикл чтения слов из файла.  
 while (true) {  
 synchronized (*lock*) {  
 // Пока не пришла очередь текущего потока или все потоки завершили работу, поток ожидает.  
 while (*currentPlayer* != threadId && !*allThreadsFinished*) {  
 *lock*.wait();  
 }  
  
 // Чтение слова из файла.  
 String word = reader.readLine();  
 if (word == null) {  
 // Если слова в файле закончились, устанавливаем флаг завершения работы всех потоков.  
 *allThreadsFinished* = true;  
 } else {  
 // Если слово прочитано, добавляем его в список текущего потока и выводим в консоль.  
 if (wordCount % *TOTAL\_THREADS* == threadId - 1) {  
 *wordArrays*[threadId - 1].add(word);  
 System.*out*.println(threadId + ": " + word);  
 }  
 wordCount++;  
 }  
  
 // Переключение на следующий поток.  
 *currentPlayer* = (*currentPlayer* % *TOTAL\_THREADS*) + 1;  
 // Уведомление всех потоков о возможности продолжить выполнение.  
 *lock*.notifyAll();  
  
 // Если все потоки завершили работу, выход из бесконечного цикла.  
 if (*allThreadsFinished*) {  
 break;  
 }  
 }  
 }  
 } catch (IOException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Основной метод программы.  
 public static void main(String[] args) throws InterruptedException {  
 // Массив потоков для выполнения игры.  
 Thread[] threads = new Thread[*TOTAL\_THREADS*];  
  
 // Создание и запуск каждого потока.  
 for (int i = 0; i < *TOTAL\_THREADS*; i++) {  
 threads[i] = new Thread(new WordGame(i + 1));  
 threads[i].start();  
 }  
  
 // Ожидание завершения работы всех потоков.  
 for (int i = 0; i < *TOTAL\_THREADS*; i++) {  
 threads[i].join();  
 }  
  
 // Вывод слов для каждого потока.  
 for (int i = 0; i < *TOTAL\_THREADS*; i++) {  
 System.*out*.println("Thread " + (i + 1) + " words: " + *wordArrays*[i]);  
 }  
  
 // Формирование списка всех слов в обратном порядке.  
 for (int i = 0; i < *WORDS\_PER\_THREAD*; i++) {  
 for (int j = 0; j < *TOTAL\_THREADS*; j++) {  
 if (*wordArrays*[j].size() > i) {  
 *allWords*.add(*wordArrays*[j].get(i));  
 }  
 }  
 }  
  
 // Вывод всех слов в обратном порядке.  
 System.*out*.println("\nВывод слов в обратном порядке:");  
  
 for (int i = *allWords*.size() - 1; i >= 0; i--) {  
 System.*out*.println(*allWords*.get(i));  
 }  
 }  
}