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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Таран

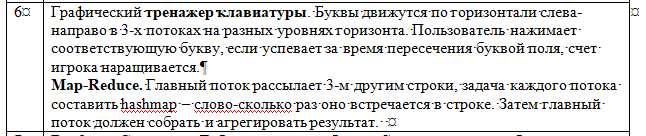
Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Тема**: Потоки в Java.

**Цель**: Освоить работу с потоками в Java.

**Задание**:

****

**Ход работы**

1. Описание работы программы тренажер клавиатуры.

Класс KeyBoardTrainer создает главное окно размером 1000x600, устанавливает прослушиватели событий, которые будут обрабатывать нажатия на клавиатуру, и создает в цикле 3 потока, каждый из которых отслеживает свою линию, на которой появляются буквы. После этого каждый поток проверяет, не успела ли “выйти” за границу буква, если нет, то удаляет ее и запрашивает перерисовку поля.

Диаграмма классов представлена на рисунке 1, а временная диаграмма работы потоков –на рисунке 2.

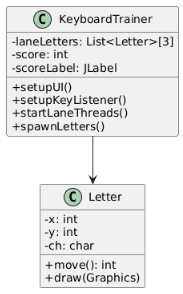


Рисунок 1 – Диаграмма классов KeyBoardTrainer

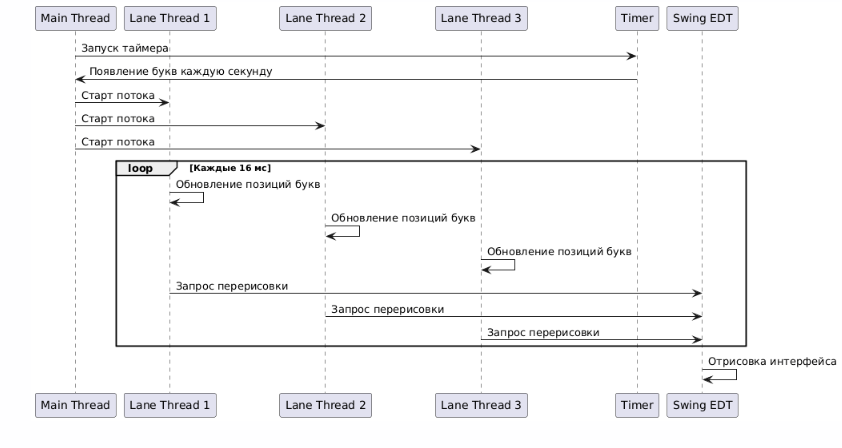


Рисунок 2 – Временная диаграмма

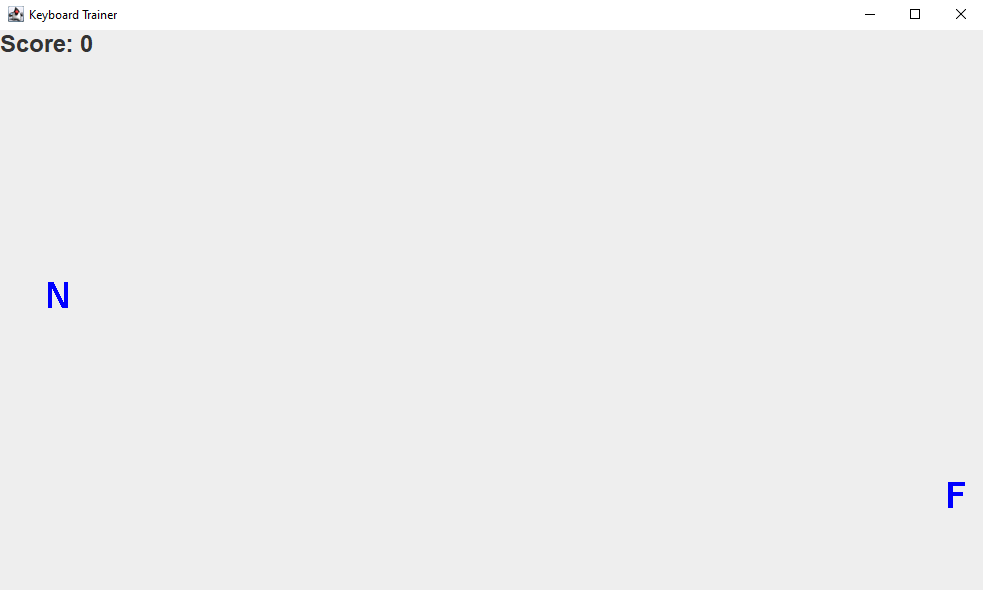


Рисунок 3 – Результат работы KeyBoarTrainer

1. Описание работы программы MapReduceThreads.

В методе main, в классе MapReduceThreads создается список с тремя строчками, на которых написаны слова. Далее создается специальная структура ConcurrentHashMap, которая предназначена для работы в многопоточной среде. Потом, для каждой строки создается отдельный поток и обработчик, который считает, сколько раз встретилось слово. Потоки запускаются и начинают вычисления, после этого результаты по очереди накапливаются в нашем HashMap с помощью join().

Диаграмма классов представлена на рисунке 4, а временная диаграмма работы потоков –на рисунке 5.

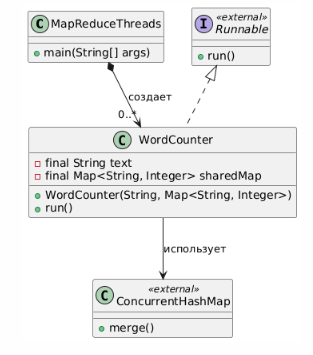
. 

Рисунок 4 – MapReduceThreads

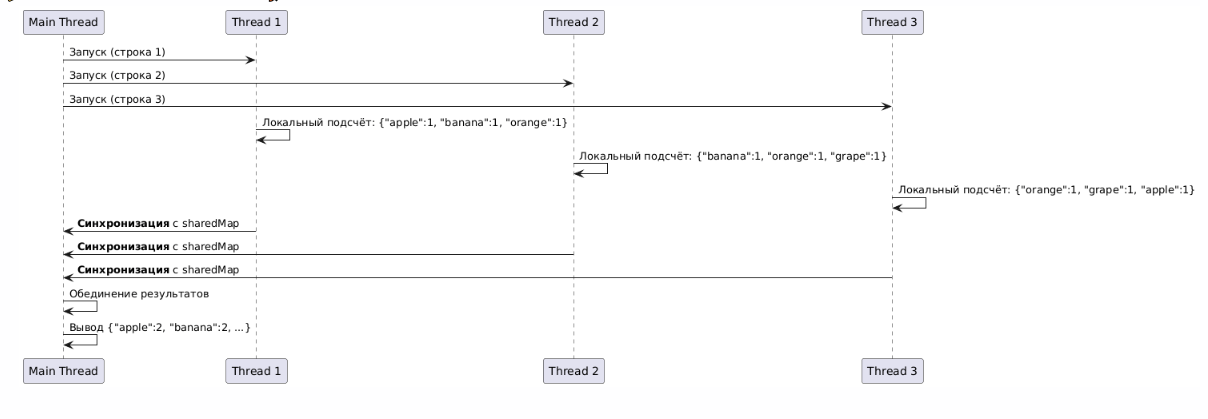
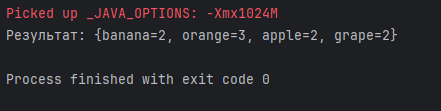


Рисунок 5 – Временная диаграмма



1. Результат работы программы MapReduceThreads

**Вывод:**В ходе выполнения работы была освоенаработа с потоками в Java.

**KeyBoardTrainer**

package ru.practicum;  
  
import javax.swing.\*;  
import javax.swing.Timer;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.\*;  
import java.util.List;  
  
public class KeyboardTrainer extends JFrame {  
  
 private static final int *WIDTH* = 1000;  
 private static final int *HEIGHT* = 600;  
 private static final int *LANES* = 3;  
 private static final int *LANE\_HEIGHT* = *HEIGHT* / *LANES*;  
 private static final int *LETTER\_SPEED* = 15;  
 private static final int *SPAWN\_RATE* = 1000;  
 private static final int *HIT\_ZONE\_START* = 0;  
 private static final Font *LETTER\_FONT* = new Font("Arial", Font.*BOLD*, 36);  
  
 private final List<Letter>[] laneLetters = new List[*LANES*];  
 private int score = 0;  
 private JLabel scoreLabel;  
  
 public KeyboardTrainer() {  
 for (int i = 0; i < *LANES*; i++) {  
 laneLetters[i] = Collections.*synchronizedList*(new ArrayList<>());  
 }  
  
 setupUI();  
 setupKeyListener();  
 startLaneThreads();  
 spawnLetters();  
 }  
  
 private void setupUI() {  
 setTitle("Keyboard Trainer");  
 setSize(*WIDTH*, *HEIGHT*);  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 scoreLabel = new JLabel("Score: 0");  
 scoreLabel.setFont(new Font("Arial", Font.*BOLD*, 24));  
 add(scoreLabel, BorderLayout.*NORTH*);  
  
 JPanel gamePanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 for (int i = 0; i < *LANES*; i++) {  
 synchronized (laneLetters[i]) {  
 laneLetters[i].forEach(letter -> letter.draw(g));  
 }  
 }  
 }  
 };  
 gamePanel.setPreferredSize(new Dimension(*WIDTH*, *HEIGHT*));  
 add(gamePanel, BorderLayout.*CENTER*);  
 }  
  
 private void setupKeyListener() {  
 addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyTyped(KeyEvent e) {  
 char pressedChar = Character.*toUpperCase*(e.getKeyChar());  
 for (int i = 0; i < *LANES*; i++) {  
 synchronized (laneLetters[i]) {  
 boolean hit = laneLetters[i].removeIf(letter ->  
 letter.getX() >= *HIT\_ZONE\_START* &&  
 letter.getChar() == pressedChar  
 );  
 if (hit) {  
 score += 10;  
 scoreLabel.setText("Score: " + score);  
 }  
 }  
 }  
 }  
 });  
 setFocusable(true);  
 requestFocusInWindow();  
 }  
  
 private void startLaneThreads() {  
 for (int i = 0; i < *LANES*; i++) {  
 final int laneIndex = i;  
 new Thread(() -> {  
 while (!Thread.*currentThread*().isInterrupted()) {  
 try {  
 synchronized (laneLetters[laneIndex]) {  
 laneLetters[laneIndex].removeIf(letter -> letter.move() > *WIDTH*);  
 }  
 SwingUtilities.*invokeLater*(this::repaint);  
 Thread.*sleep*(16);  
 } catch (InterruptedException ex) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }).start();  
 }  
 }  
  
 private void spawnLetters() {  
 new Timer(*SPAWN\_RATE*, e -> {  
 Random rand = new Random();  
 char c = (char) ('A' + rand.nextInt(26));  
 int lane = rand.nextInt(*LANES*);  
 int y = lane \* *LANE\_HEIGHT* + 50;  
 synchronized (laneLetters[lane]) {  
 laneLetters[lane].add(new Letter(c, y));  
 }  
 }).start();  
 }  
  
 private static class Letter {  
 private int x = 0;  
 private final int y;  
 private final char ch;  
  
 public Letter(char ch, int y) {  
 this.ch = ch;  
 this.y = y;  
 }  
  
 public int move() {  
 return x += *LETTER\_SPEED*;  
 }  
  
 public char getChar() {  
 return ch;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public void draw(Graphics g) {  
 g.setColor(Color.*BLUE*);  
 g.setFont(*LETTER\_FONT*);  
 g.drawString(String.*valueOf*(ch), x, y);  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> new KeyboardTrainer().setVisible(true));  
 }  
}

}

**MapReduceThreads**

package ru.practicum;  
  
import java.util.\*;  
import java.util.concurrent.ConcurrentHashMap;  
  
public class MapReduceThreads {  
  
 public static void main(String[] args) throws InterruptedException {  
 List<String> data = Arrays.*asList*(  
 "apple banana orange",  
 "banana orange grape",  
 "orange grape apple"  
 );  
  
 List<Thread> threads = new ArrayList<>();  
 Map<String, Integer> resultMap = new ConcurrentHashMap<>();  
  
 for (String line : data) {  
 WordCounter counter = new WordCounter(line, resultMap);  
 Thread thread = new Thread(counter);  
 threads.add(thread);  
 thread.start();  
 }  
  
 for (Thread thread : threads) {  
 thread.join();  
 }  
  
 System.*out*.println("Результат: " + resultMap);  
 }  
  
 static class WordCounter implements Runnable {  
 private final String text;  
 private final Map<String, Integer> sharedMap;  
  
 WordCounter(String text, Map<String, Integer> sharedMap) {  
 this.text = text;  
 this.sharedMap = sharedMap;  
 }  
  
 @Override  
 public void run() {  
 Map<String, Integer> localMap = new HashMap<>();  
 String[] words = text.split("\\s+");  
  
 for (String word : words) {  
 localMap.put(word, localMap.getOrDefault(word, 0) + 1);  
 }  
  
 synchronized (sharedMap) {  
 localMap.forEach((word, count) ->  
 sharedMap.merge(word, count, Integer::*sum*)  
 );  
 }  
 }  
 }  
}