**Лабораторная работа №4**

**Текстовые файлы и исключения**

Дан текстовый файл, каждая строка которого имеет предложенный ниже формат. Необходимо вывести его данные в виде таблицы, и выполнить указанную обработку. Имя файла задается пользователем. Если выбранный файл имеет иной формат, то сообщить об этом, используя механизм исключений.

В качестве интерфейса для ввода исходных данных и вывода результатов использовать web-интерфейс организованный с помощью фреймворка Vaadin. Для выбора операции организовать вертикальное меню, для расположения компонентов использовать **GridLayout**, у страницы должен быть заголовок, например, «Лабораторная работа №4» (стиль заголовка выбрать самостоятельно, заголовок поместить в одну из ячеек **GridLayout**).

Данные из файла считать двумя способами:

1. Стандартное чтение из потока с помощью классов **FileReader** или **Scanner**;

2. Чтение из потока в функциональном стиле, используя метод **lines()** класса **Files**.

Сообщение об исключительной ситуации также отразить на странице.

Вариант 9. Школа. Найти суммарную нагрузку всех учителей школы, а также по названию предмета, определить, кто из учителей его преподает.

Формат строки:

Фамилия учителя (строка из 20 символов) предмет (строка из 20 символов) количество часов в неделю (целое число).

**Теоретическая часть**

Исключение (exception) – это событие, возникающее во время выполнения программы, которое нарушает нормальный ход выполнения инструкций. Исключения используются для обработки ошибок и других неожиданных ситуаций.

Механизм обработки исключений позволяет:

* Перехватывать исключения (**try-catch**)
* Пробрасывать их дальше (**throws**)
* Создавать свои исключения (**throw**)

**Текст программы**

**Teacher.java**:

package mypack;  
  
public class Teacher {  
 private String name; *// фамилия* private String subject; *// предмет* private int hours; *// количество часов* public Teacher(String name, String subject, int hours) {  
 this.name = name.trim();  
 this.subject = subject.trim();  
 this.hours = hours;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getSubject() {  
 return subject;  
 }  
  
 public int getHours() {  
 return hours;  
 }  
}

**MainView.java**:

package com.northpole.snow;  
  
import com.vaadin.flow.component.Component;  
import com.vaadin.flow.component.button.Button;  
import com.vaadin.flow.component.combobox.ComboBox;  
import com.vaadin.flow.component.grid.Grid;  
import com.vaadin.flow.component.grid.HeaderRow;  
import com.vaadin.flow.component.html.H1;  
import com.vaadin.flow.component.html.Paragraph;  
import com.vaadin.flow.component.orderedlayout.HorizontalLayout;  
import com.vaadin.flow.component.orderedlayout.VerticalLayout;  
import com.vaadin.flow.component.tabs.Tab;  
import com.vaadin.flow.component.tabs.Tabs;  
import com.vaadin.flow.component.textfield.TextField;  
import com.vaadin.flow.component.upload.Upload;  
import com.vaadin.flow.component.upload.receivers.MemoryBuffer;  
import com.vaadin.flow.router.Route;  
import mypack.Teacher;  
  
import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.util.\*;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
@Route  
public class MainView extends VerticalLayout {  
  
 private List<Teacher> teachers = new ArrayList<>(); *// Список преподавателей* private Grid<Teacher> grid = new Grid<>(Teacher.class, false); *// Таблица для отображения данных* private Paragraph errorMsg = new Paragraph(); *// Сообщение об ошибках* private Paragraph summary = new Paragraph(); *// Блок с суммарной нагрузкой* private ComboBox<String> methodSelect; *// Выбор метода чтения файла* public MainView() {  
 setPadding(true); *// Настройка отступов* setSpacing(true); *// Настройка промежутков между компонентами* setSizeFull(); *// Занимает все доступное пространство  
  
 // Создание заголовка* H1 title = new H1("Лабораторная работа №4");  
  
 *// Настройка выбора метода чтения файла* methodSelect = new ComboBox<>("Способ чтения файла");  
 methodSelect.setItems("Scanner", "Files.lines()");  
 methodSelect.setValue("Scanner");  
  
 *// Настройка компонента загрузки файлов* MemoryBuffer buffer = new MemoryBuffer();  
 Upload upload = new Upload(buffer);  
 upload.setAcceptedFileTypes(".txt"); *// Только txt файлы* upload.setMaxFiles(1); *// Максимум 1 файл  
  
 // Обработчик успешной загрузки файла* upload.addSucceededListener(event -> {  
 try (InputStream is = buffer.getInputStream()) {  
 if ("Scanner".equals(methodSelect.getValue())) {  
 readWithScanner(is); *// Чтение через Scanner* } else {  
 readWithFilesLines(is); *// Чтение через Files.lines()* }  
 updateGrid(); *// Обновление таблицы* errorMsg.setText(""); *// Очистка сообщения об ошибке* } catch (Exception ex) {  
 teachers.clear();  
 updateGrid();  
 errorMsg.setText("Ошибка чтения файла: " + ex.getMessage());  
 }  
 });  
  
 *// Настройка колонок таблицы* Grid.Column<Teacher> nameCol = grid.addColumn(Teacher::getName).setHeader("Фамилия").setKey("name");  
 Grid.Column<Teacher> subjectCol = grid.addColumn(Teacher::getSubject).setHeader("Предмет").setKey("subject");  
 Grid.Column<Teacher> hoursCol = grid.addColumn(Teacher::getHours).setHeader("Часы").setKey("hours");  
  
 *// Добавление заголовка таблицы* HeaderRow topHeader = grid.prependHeaderRow();  
 topHeader.join(  
 topHeader.getCell(nameCol),  
 topHeader.getCell(subjectCol),  
 topHeader.getCell(hoursCol)  
 ).setText("Лабораторная работа 4");  
  
 *// Настройка вкладок* Map<Tab, Component> tabContent = new LinkedHashMap<>();  
 Tab tableTab = new Tab("Таблица");  
 Tab summaryTab = new Tab("Нагрузка");  
 Tab queryTab = new Tab("Поиск по предмету");  
  
 tabContent.put(tableTab, grid);  
 tabContent.put(summaryTab, summary);  
  
 *// Настройка панели поиска по предмету* VerticalLayout subjectSearch = new VerticalLayout();  
 TextField subjectField = new TextField("Название предмета");  
 Paragraph result = new Paragraph();  
 Button searchBtn = new Button("Найти");  
 searchBtn.addClickListener(e -> {  
 String subject = subjectField.getValue().trim().toLowerCase();  
 List<String> matches = teachers.stream()  
 .filter(t -> t.getSubject().equalsIgnoreCase(subject))  
 .map(Teacher::getName)  
 .distinct()  
 .collect(Collectors.*toList*());  
 if (matches.isEmpty()) {  
 result.setText("Учителей по предмету не найдено.");  
 } else {  
 result.setText("Преподают: " + String.*join*(", ", matches));  
 }  
 });  
 subjectSearch.add(subjectField, searchBtn, result);  
 tabContent.put(queryTab, subjectSearch);  
  
 *// Настройка вертикальных вкладок* Tabs tabs = new Tabs(tableTab, summaryTab, queryTab);  
 tabs.setOrientation(Tabs.Orientation.*VERTICAL*);  
  
 *// Контейнер для содержимого вкладок* VerticalLayout pages = new VerticalLayout();  
 tabContent.values().forEach(pages::add);  
 tabContent.values().forEach(c -> c.setVisible(false));  
 grid.setVisible(true); *// По умолчанию показываем таблицу  
  
 // Обработчик переключения вкладок* tabs.addSelectedChangeListener(event -> {  
 tabContent.values().forEach(c -> c.setVisible(false));  
 Component selected = tabContent.get(tabs.getSelectedTab());  
 if (selected != null) selected.setVisible(true);  
 });  
  
 pages.setSizeFull();  
  
 *// Горизонтальный макет с вкладками и содержимым* HorizontalLayout tabsWithContent = new HorizontalLayout(tabs, pages);  
 tabsWithContent.setSizeFull();  
 tabsWithContent.setFlexGrow(0, tabs);  
 tabsWithContent.setFlexGrow(1, pages);  
  
 errorMsg.getStyle().set("color", "red"); *// Красный цвет для ошибок  
  
 // Добавление всех компонентов на форму* add(title, methodSelect, upload, tabsWithContent, errorMsg);  
 setFlexGrow(1, tabsWithContent);  
 }  
  
 *// Обновление данных в таблице и расчет суммарной нагрузки* private void updateGrid() {  
 grid.setItems(teachers);  
 int totalHours = teachers.stream().mapToInt(Teacher::getHours).sum();  
 summary.setText("Суммарная нагрузка всех учителей: " + totalHours + " часов");  
 }  
  
 *// Парсинг строки с данными преподавателя* private Teacher parseLine(String line) {  
 if (line.length() < 40)  
 throw new IllegalArgumentException("Неверный формат строки: строка слишком короткая");  
 String name = line.substring(0, 20).trim(); *// Фамилия (первые 20 символов)* String subject = line.substring(20, 40).trim(); *// Предмет (следующие 20 символов)* String hoursStr = line.length() > 40 ? line.substring(40).trim() : "0"; *// Часы (остаток строки)* int hours;  
 try {  
 hours = Integer.*parseInt*(hoursStr);  
 } catch (NumberFormatException e) {  
 throw new IllegalArgumentException("Неверный формат часов в строке");  
 }  
 return new Teacher(name, subject, hours);  
 }  
  
 *// Чтение файла с помощью Scanner* private void readWithScanner(InputStream is) throws IOException {  
 teachers.clear();  
  
 File tempFile = File.*createTempFile*("upload", ".txt");  
 tempFile.deleteOnExit();  
  
 try (OutputStream os = new FileOutputStream(tempFile)) {  
 is.transferTo(os);  
 }  
  
 try (Scanner scanner = new Scanner(new FileReader(tempFile, StandardCharsets.*UTF\_8*))) {  
 while (scanner.hasNextLine()) {  
 String line = scanner.nextLine();  
 teachers.add(parseLine(line));  
 }  
 }  
 }  
  
 *// Чтение файла с помощью Files.lines()* private void readWithFilesLines(InputStream is) throws IOException {  
 teachers.clear();  
  
 File tempFile = File.*createTempFile*("upload", ".txt");  
 tempFile.deleteOnExit();  
  
 try (OutputStream os = new FileOutputStream(tempFile)) {  
 is.transferTo(os);  
 }  
  
 try (Stream<String> lines = Files.*lines*(tempFile.toPath(), StandardCharsets.*UTF\_8*)) {  
 teachers = lines.map(this::parseLine)  
 .collect(Collectors.*toList*());  
 }  
 }  
}

**Описание программы**

Класс **Teacher**:

* **name** – фамилия учителя
* **subject** – предмет
* **hours** – количество часов
* **Teacher**() – конструктор
* **getName**() – геттер для имени
* **getSubject**() – геттер для предмета
* **getHours**() – геттер для количества часов

Класс **MainView**:

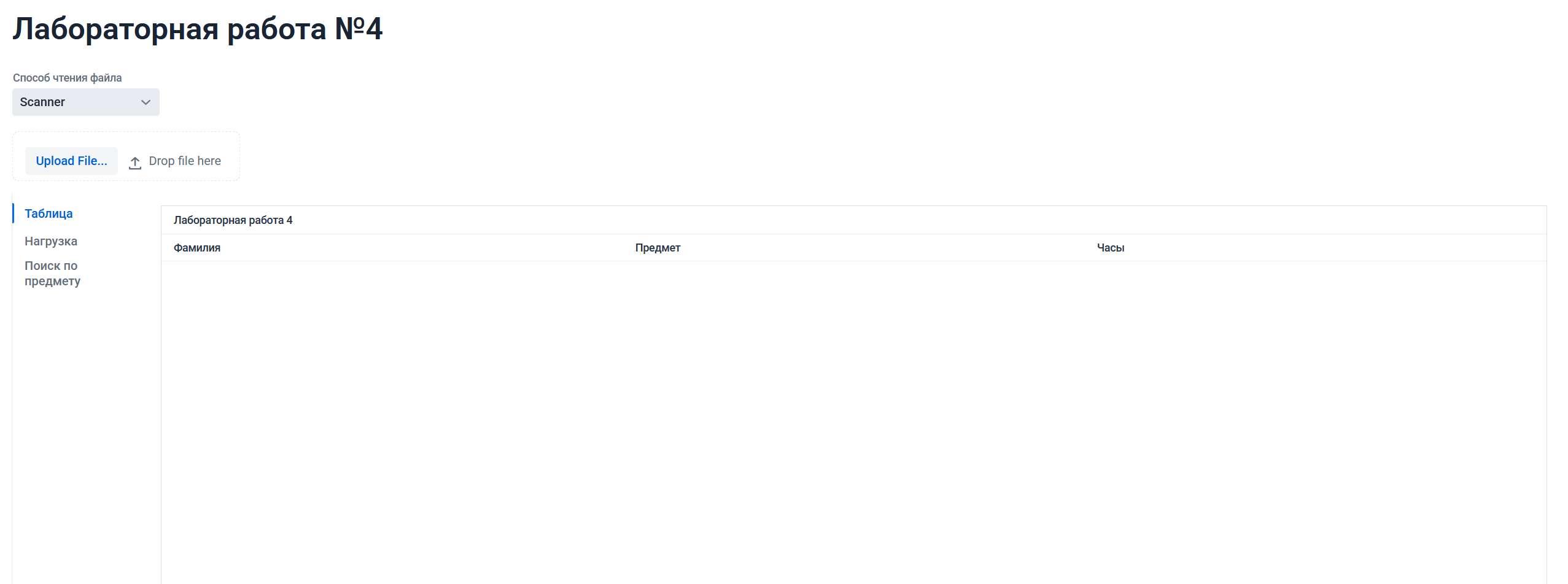
* **teachers** – список всех учителей, считанных из файла
* **grid** – таблица для отображения данных
* **errorMsg** – поле для вывода ошибок
* **summary** – текст для отображения суммарной нагрузки
* **methodSelect** – выпадающий список для выбора способа чтения файла
* **MainView**() – конструктор. В конструкторе настраивается интерфейс пользователя (заголовок, ComboBox, компонент загрузки файла, таблица, табы, панель поиска по предмету, система вкладок, размещение на экране)
* **updateGrid**() – обновляет **grid** новым списком **teachers** и считает общую нагрузку **totalHours**
* **parseLine**() – получает данные из одной строки файла: символы с 0 по 19 – фамилия, с 20 по 39 – предмет, с 40 и дальше – часы. Обрезает пробелы, преобразует количество часов в целое число. При неправильном формате выбрасывает исключение
* **readWithScanner**() – создает временный файл, считывает строки через Scanner. Каждая строка преобразуется в объект класса **Teacher** и добавляется в список
* **readWithFilesLines**() – аналогично предыдущему, но считывание строк через Files.lines()

**Исключения**

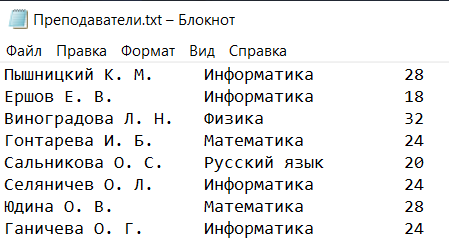
|  |  |  |
| --- | --- | --- |
| Исключение | Метод | Причина |
| **IllegalArgumentException** | **parseLine()** | Строка короче 40 символов |
| **NumberFormatException** | **parseLine()**, при использовании **Integer.parseInt()** | В поле часов – не число (например, буквы или пустая строка) |
| **IOException** | **readWithScanner()** и **readWithFilesLines()** | Ошибка создания или чтения временного файла |
| **Exception** | В блоке **catch (Exception ex)** при загрузке файла | Перехватывает любые ошибки, включая неожиданные |

**Результаты тестов**

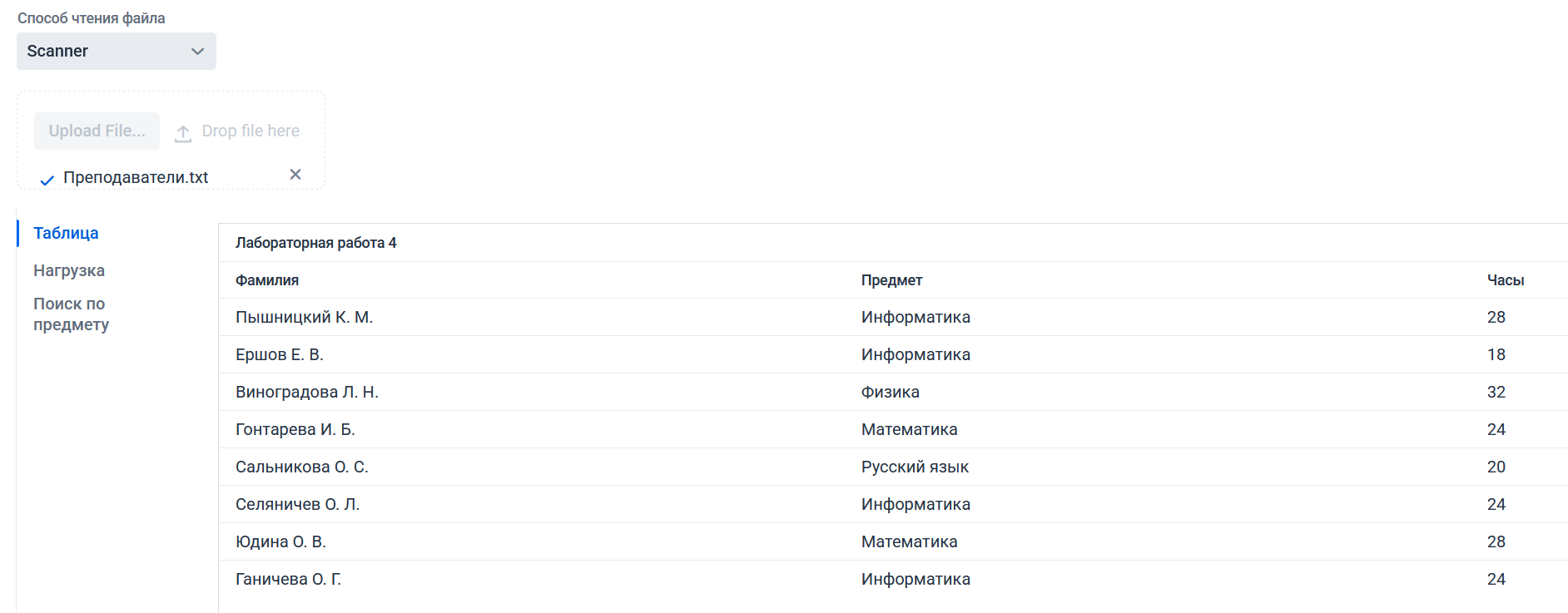
Программа до загрузки файла:



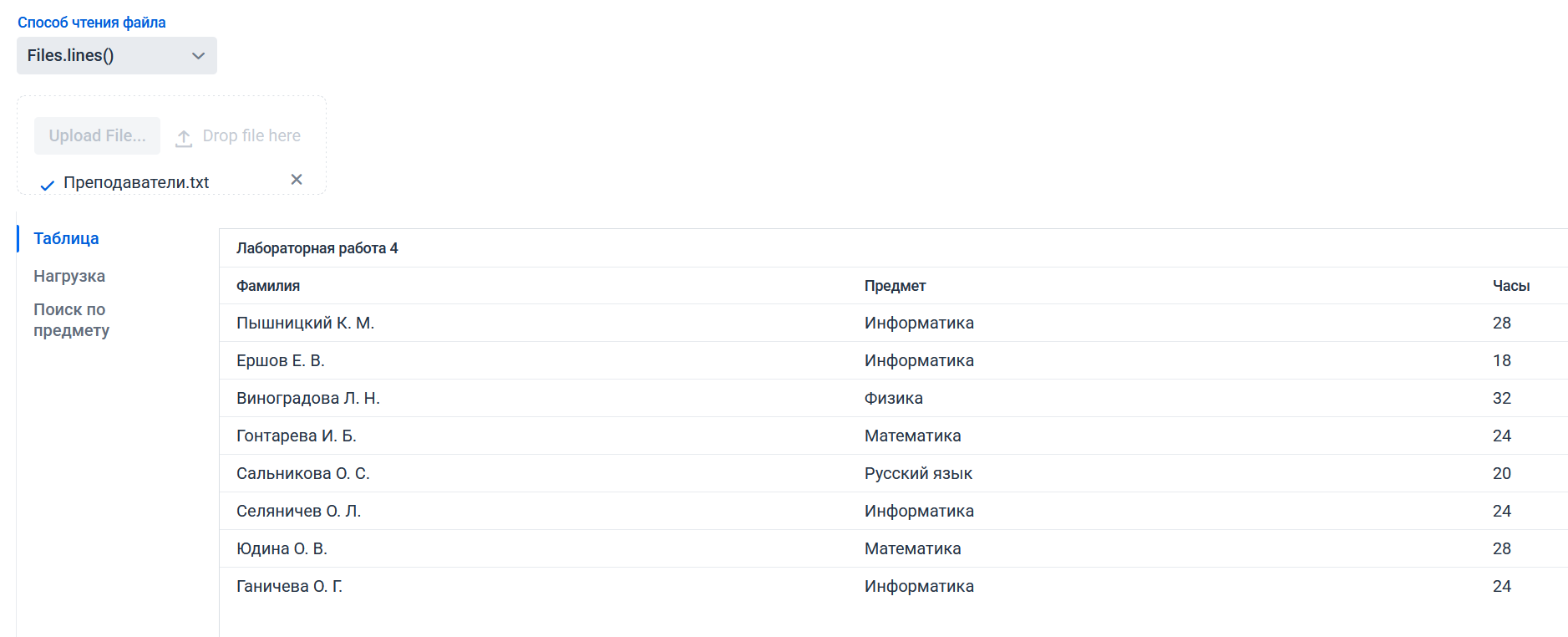
Исходный файл:



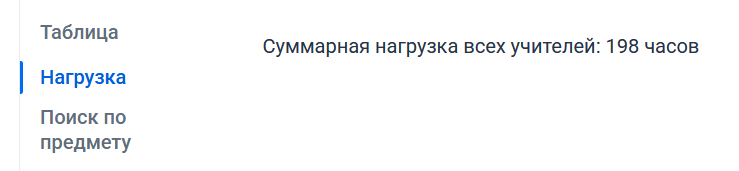
Чтение с помощью **Scanner**:



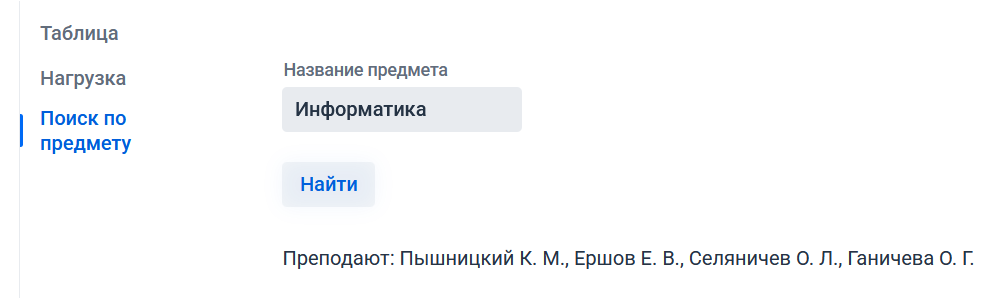
Чтение с помощью **Files.lines**:

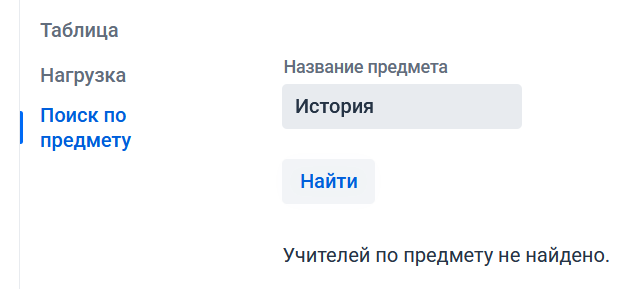


Общая нагрузка:

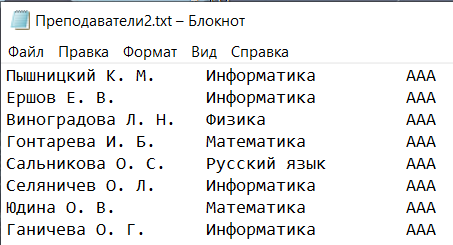


Поиск по предмету:

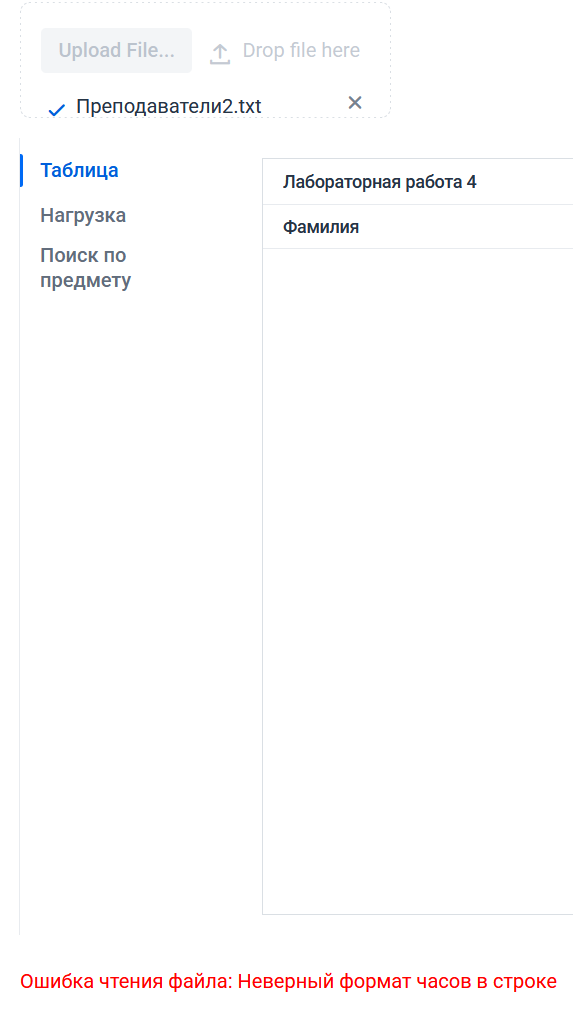




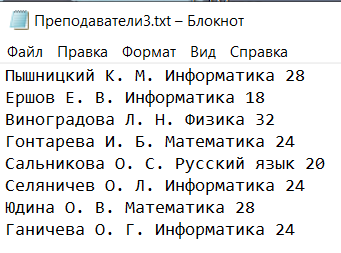
Файл с неверным форматом часов:



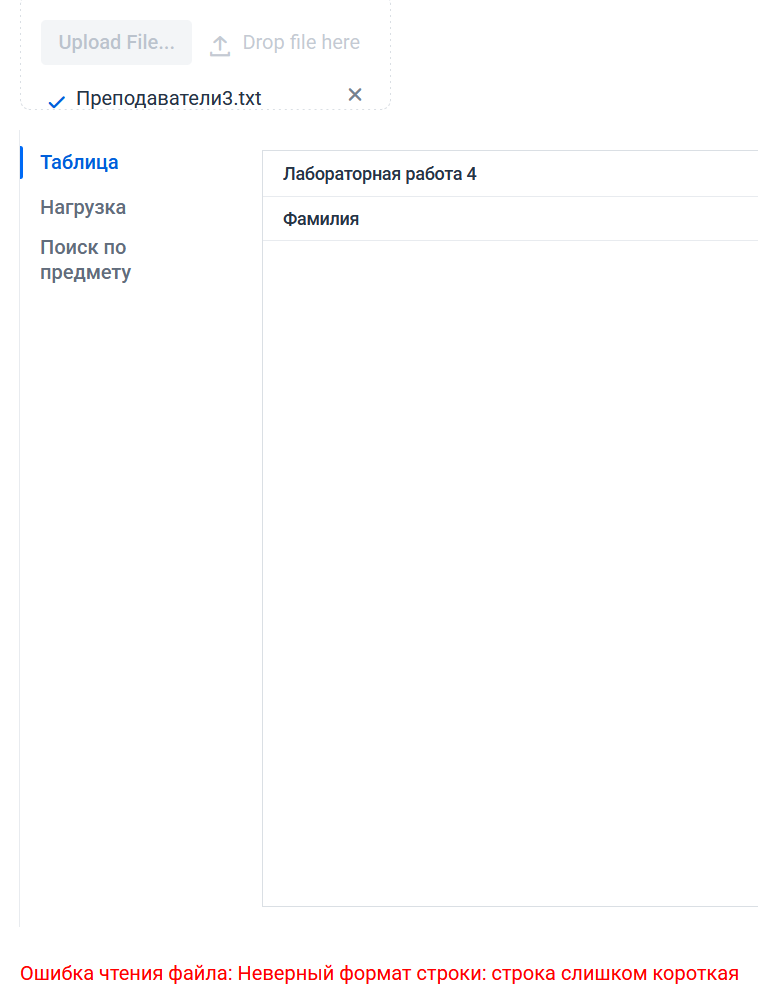
Сообщение об ошибке:



Файл с неверным форматом строк:



Сообщение об ошибке:



**Вывод**

В ходе выполнения лабораторной работы была создана программа, которая обрабатывает файл со списком преподавателей, выводя строки с данными в таблицу. Также программа подсчитывает общее количество часов и позволяет искать преподавателей по названию предмета. Интерфейс программы выполнен с помощью фреймворка Vaadin.