**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**ИДЗ**

**ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ KOTLIN»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Новиков Г.В. |
| Студентка гр. 1302 |  | Романова О.В. |
| Преподаватель |  | Кулагин М.В. |

Санкт-Петербург

2023

# Разработка приложения для взаимодействия с внешними источниками

# 1.1. Задание

Напишите программу, которая с консоли считывает поисковый запрос пользователя, и выводит результат поиска по Википедии. После выбора нужной статьи программа должна открывать ее в браузере. Программа должна реагировать корректно на любой пользовательский ввод.

Задача разбивается на 5 этапов:

1. Считать введенные пользователем данные
2. Сделать запрос к серверу
3. Распарсить ответ
4. Вывести результат поиска
5. Открыть нужную страницу в браузере

Использовать готовые библиотеки для работы с Википедией нельзя.

# 1.2. Спецификация программы

Описание реализованных классов. UML-диаграмма классов.

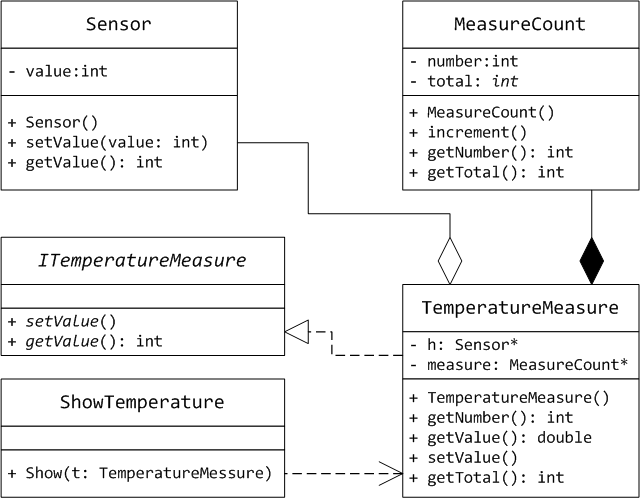


Рисунок 1. Диаграмма классов

# 1.3. Описание интерфейса пользователя программы

Пользователь вводит запрос для поиска по Википедии (рис.2).



Рис. 2

В консоль выводятся результаты поиска – заголовки и json (рис.3).

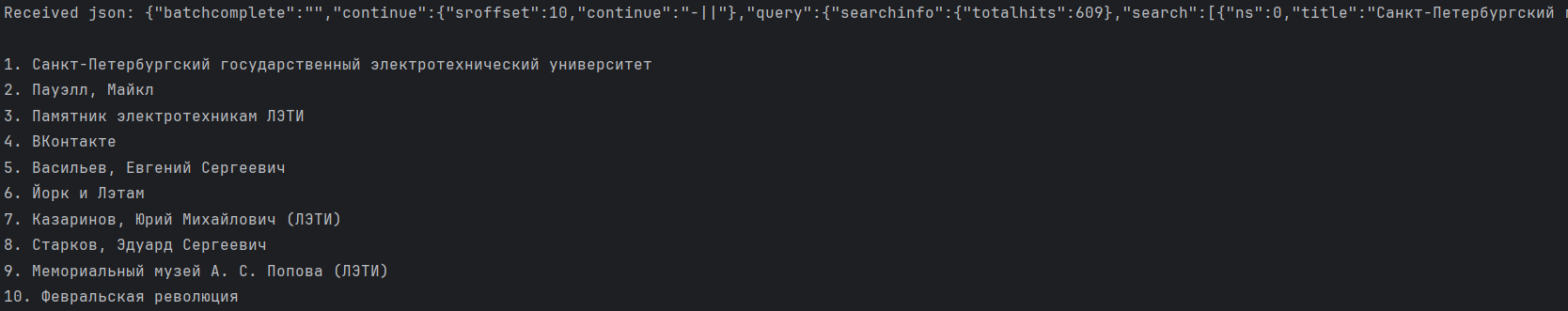


Рис. 3

Пользователь вводит номер заголовка, который необходимо открыть (рис.4).



Рис. 4

В консоль выводится URL и автоматически открывается ссылка (рис.5).



Рис. 5

# 1.4. Текст программы

Main.java:

import java.awt.Desktop;  
import java.io.IOException;  
import java.net.URISyntaxException;  
import java.util.Scanner;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonArray;  
import java.net.URI;  
import java.util.regex.Pattern;  
  
public class Main  
{  
 private static String ReadWordFromConsole(Scanner in) {  
 String request;  
 while (true) {  
 System.*out*.print("Enter word: ");  
 request = in.nextLine();  
 if (request.isBlank()) {  
 System.*out*.println("Please enter word to search\n");  
 } else break;  
 }  
 return request;  
 }  
  
 public static int ReadArticleNumber(Scanner in, int max) {  
 String choice\_str;  
 int choice;  
  
 while (true) {  
 System.*out*.print("\nEnter article number: ");  
  
 choice\_str = in.nextLine().strip();  
 Pattern pattern = Pattern.*compile*("\\d+");  
 if (!choice\_str.isBlank() && pattern.matcher(choice\_str).matches()) {  
 choice = Integer.*parseInt*(choice\_str);  
 if (0 < choice && choice <= max) break;  
 }  
 System.*out*.println("Please enter number from 1 to " + max);  
 }  
 return choice;  
 }  
  
 public static void main(String[] args) throws IOException, URISyntaxException, InterruptedException {  
 Scanner in = new Scanner(System.*in*);  
  
 // Search and send request to the server  
 SearchEngine engine = new SearchEngine();  
 String result\_json;  
  
 String request;  
 while (true) {  
 try {  
 request = *ReadWordFromConsole*(in);  
 result\_json = engine.WikiSearch(request);  
 break;  
 } catch (IOException e) {  
 System.*out*.println("Error: cannot connect to Wikipedia. Please check your internet connection");  
 }  
 }  
  
 System.*out*.println("Received json: " + result\_json + "\n");  
  
 // Parsing  
 GsonBuilder builder = new GsonBuilder();  
 Gson gson = builder.create();  
  
 JsonObject json = gson.fromJson(result\_json, JsonObject.class);  
 JsonObject query = json.getAsJsonObject("query");  
 JsonArray result = query.getAsJsonArray("search");  
  
 if (result.isEmpty())  
 {  
 System.*out*.println("No results found");  
 System.*exit*(0);  
 }  
 // Titles output  
 String title;  
 for (int i = 0; i < result.size(); i++)  
 {  
 JsonObject titles = result.get(i).getAsJsonObject();  
 title = titles.get("title").getAsString();  
 System.*out*.println((i + 1) + ". " + title);  
 }  
  
 int choice = *ReadArticleNumber*(in, result.size());  
  
 // Getting pageId  
 JsonObject id = result.get(choice - 1).getAsJsonObject();  
 String pageId = id.get("pageid").getAsString();  
  
 // Open wiki  
 String wiki\_url = "https://ru.wikipedia.org/w/index.php?curid=" + pageId;  
 System.*out*.println("URL:" + wiki\_url);  
 Desktop.*getDesktop*().browse(new URI(wiki\_url));  
  
 in.close();  
 }  
}

SearchEngine.java:

import java.io.IOException;  
import java.net.URI;  
import java.net.URISyntaxException;  
import java.net.URLEncoder;  
import java.net.http.HttpClient;  
import java.net.http.HttpRequest;  
import java.net.http.HttpResponse;  
import java.net.http.HttpResponse.BodyHandlers;  
import java.nio.charset.StandardCharsets;  
  
public class SearchEngine  
{  
 public String WikiSearch(String requestString) throws IOException, InterruptedException {  
 URI uri;  
 try  
 {  
 uri = new URI("https://ru.wikipedia.org/w/api.php?action=query&list=search&utf8=&format=json&srsearch=" + URLEncoder.*encode*(requestString, StandardCharsets.*UTF\_8*));  
 } catch (URISyntaxException e)  
 {  
 throw new RuntimeException(e);  
 }  
 HttpRequest request = HttpRequest.*newBuilder*(uri).build();  
  
 HttpClient client = HttpClient.*newBuilder*()  
 .version(HttpClient.Version.*HTTP\_2*)  
 .build();  
  
 HttpResponse<String> response = client.send(request, BodyHandlers.*ofString*());  
  
 return response.body();  
 }  
}

# 1.5. Выводы

В ходе выполнения лабораторной работы мы впервые поработали с Java, познакомились с библиотекой Gson и классом Desktop. Также в лабораторной работе использовались API, JSON и URL. Библиотека Gson используется для преобразования JSON, класс Desktop используется для автоматического открытия статьи Википедии в Google.

# Разработка приложения для обработки данных

# 2.1. Задание

Даны 2 файла-справочника городов. Один файл в формате xml, другой в формате csv.

Необходимо разработать консольное приложение для работы с ними.

После запуска приложение ожидает ввода пути до файла-справочника либо команды на завершение работы (какая-то комбинация клавиш).

По команде завершения работы приложение завершает свою работу.

После ввода пути до файла-справочника приложение формирует сводную статистику:

1) Отображает дублирующиеся записи с количеством повторений.

2) Отображает, сколько в каждом городе: 1, 2, 3, 4 и 5-этажных зданий.

3) Показывает время обработки файла.

После вывода статистики приложение снова ожидает ввода пути до файла-справочника либо команды на завершение работы.

В процессе работы приложение падать не должно, выход только по команде на завершение работы.

# 2.2. Спецификация программы

Описание реализованных классов. UML-диаграмма классов.

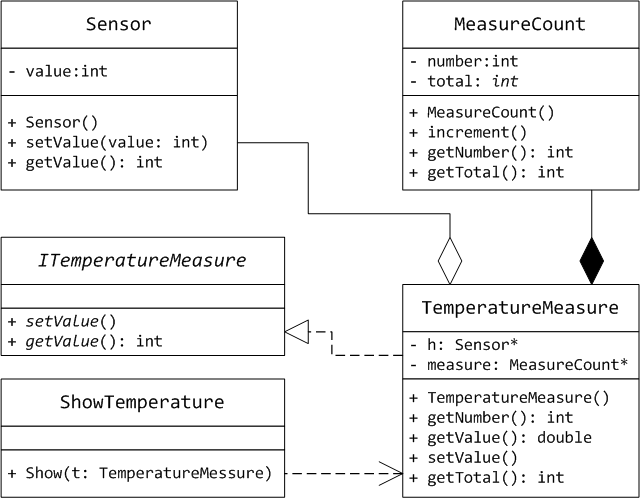


Рисунок 1. Диаграмма классов

# 2.3. Описание интерфейса пользователя программы

Пользователь вводит запрос: адрес файла или слово для выхода (рис.7).

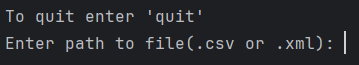


Рис. 7

При вводе адреса в консоль выводится результат обработки файла (дубликаты и количество 1, 2, 3, 4 и 5-этажных зданий) и время обработки файла (рис.8). После этого снова выводится запрос на адрес или выход (рис.9).

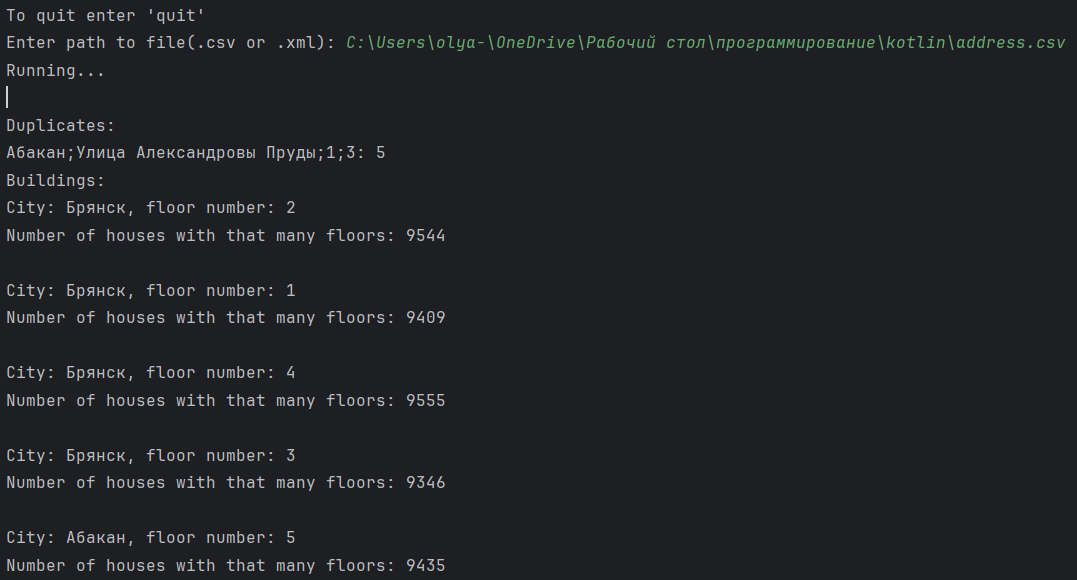


Рис. 8 – дубликаты и количество 1, 2, 3, 4 и 5-этажных зданий

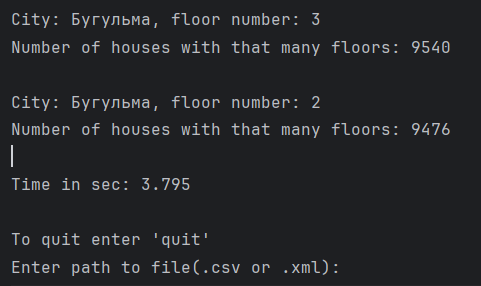


Рис. 9 – время обработки и новый запрос

При вводе слова “quit” программа завершается (рис.10).

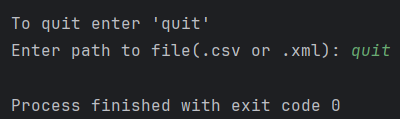


Рис. 10

# 2.4. Текст программы

Main.java:

public class Main {  
 public static void main(String[] args) {  
 IOHandler ioHandler = new IOHandler();  
 ioHandler.run();  
 }  
}

Address.java:

public class Address {  
 private String address;  
 public String getAddress() {  
 return address;  
 }  
 public void setAddress(String address) {  
 this.address = address;  
 }  
}

AddressList.java:

import java.util.HashMap;  
  
public class AddressList {  
 public HashMap<String, Integer> addresses = new HashMap<>();  
 public HashMap<String, Integer> buildingsCount = new HashMap<>();  
 public HashMap<String, Integer> duplicates = new HashMap<>();  
  
 public void addAddress(String address) {  
 updateBuildingsCount(address);  
  
 if (addresses.containsKey(address)) {  
 addresses.put(address, addresses.get(address) + 1);  
 duplicates.put(address, addresses.get(address));  
 }  
 else {  
 addresses.put(address, 1);  
 }  
 }  
  
 private void updateBuildingsCount(String address) {  
 String key = address.substring(0, address.indexOf(";")) +  
 address.substring(address.lastIndexOf(";"));  
 buildingsCount.put(key, buildingsCount.getOrDefault(key, 0) + 1);  
 }  
}

CsvParser.java:

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
  
public class CsvParser extends Parser {  
 public String requiredExtension = "csv";  
  
 @Override  
 public AddressList parse(String fileName) throws IllegalArgumentException, IOException {  
 if (fileExtensionIsInvalid(fileName, requiredExtension)) {  
 throw new IllegalArgumentException();  
 }  
  
 AddressList addresses = new AddressList();  
 BufferedReader reader = new BufferedReader(new FileReader(fileName));  
  
 String line;  
 line = reader.readLine();  
  
 while ((line = reader.readLine()) != null) {  
 addresses.addAddress(line.replaceAll("\"", ""));  
 }  
  
 return addresses;  
 }  
}

IOHandler.java:

import java.io.\*;  
import java.util.Scanner;  
  
public class IOHandler {  
  
 */\*\* Starts loop that takes from user a path to scv or xml file, analyses it and writes result to console  
 'quit' exits the loop \*/* public void run() {  
 AddressList addresses;  
  
 String input = getUserInput();  
 while (!input.equals("quit")) {  
 System.*out*.println("Running...\n");  
 try {  
 long startTime = System.*currentTimeMillis*();  
  
 addresses = readFile(input);  
 printResult(addresses);  
  
 long endTime = System.*currentTimeMillis*();  
 long time = endTime - startTime;  
 System.*out*.println("Time in sec: " + (time \* 0.001));  
 } catch (IllegalArgumentException e) {  
 System.*out*.println("Illegal filename");  
 } catch (IOException e) {  
 System.*out*.println("File not found");  
 }  
  
 input = getUserInput();  
 }  
 }  
  
 private String getUserInput()  
 {  
 System.*out*.print("\nTo quit enter 'quit'\n");  
 System.*out*.print("Enter path to file(.csv or .xml): ");  
  
 Scanner in = new Scanner(System.*in*);  
  
 return in.nextLine();  
 }  
  
 private AddressList readFile(String fileName) throws IllegalArgumentException, IOException {  
 int i = fileName.lastIndexOf('.');  
 if (i == -1) {  
 throw new IllegalArgumentException();  
 }  
 String extension = fileName.substring(i + 1);  
  
 Parser parser = switch (extension) {  
 case "csv" -> new CsvParser();  
 case "xml" -> new XmlParser();  
 default -> throw new IllegalArgumentException();  
 };  
  
 return parser.parse(fileName);  
 }  
  
 private void printResult(AddressList addresses) {  
 System.*out*.println("Duplicates:");  
 for (String address : addresses.duplicates.keySet()) {  
 System.*out*.println(address + ": " + addresses.duplicates.get(address));  
 }  
 System.*out*.println("Buildings:");  
 for (String key : addresses.buildingsCount.keySet()) {  
 String city = key.split(";")[0];  
 String floor = key.split(";")[1];  
 System.*out*.println("City: " + city + ", floor number: " + floor);  
 System.*out*.println("Number of houses with that many floors: " + addresses.buildingsCount.get(key) + "\n");  
 }  
 }  
}

Parser.java:

import java.io.IOException;  
  
public abstract class Parser {  
 public abstract AddressList parse(String fileName) throws IllegalArgumentException, IOException;  
  
 protected boolean fileExtensionIsInvalid(String fileName, String requiredExtension) {  
 int i = fileName.lastIndexOf('.');  
 String extension;  
 if (i != -1) {  
 extension = fileName.substring(i + 1);  
 } else {  
 return true;  
 }  
  
 return !extension.equals(requiredExtension);  
 }  
}

SaxAddressParser.java:

import org.xml.sax.Attributes;  
import org.xml.sax.helpers.DefaultHandler;  
  
import java.util.ArrayList;  
  
public class SaxAddressParser extends DefaultHandler {  
 private final ArrayList<Address> addressList = new ArrayList<>();  
 private Address address = null;  
  
 public ArrayList<Address> getAddressList() {  
 return addressList;  
 }  
  
 @Override  
 public void startElement(String uri, String localName, String qName, Attributes attributes) {  
 if (qName.equals("item")) {  
 address = new Address();  
 String city = attributes.getValue("city");  
 String street = attributes.getValue("street");  
 String house = attributes.getValue("house");  
 String floor = attributes.getValue("floor");  
  
 String sb = city + ";" + street + ";" + house + ";" + floor;  
 address.setAddress(sb);  
 }  
 }  
  
 @Override  
 public void endElement(String uri, String localName, String qName) {  
 if (qName.equals("item")) {  
 addressList.add(address);  
 }  
 }  
}

XmlParser.java:

import org.xml.sax.SAXException;  
  
import javax.xml.parsers.\*;  
import java.io.File;  
import java.io.IOException;  
import java.util.ArrayList;  
  
public class XmlParser extends Parser {  
 public String requiredExtension = "xml";  
  
 @Override  
 public AddressList parse(String fileName) throws IllegalArgumentException, IOException {  
 if (fileExtensionIsInvalid(fileName, requiredExtension)) {  
 throw new IllegalArgumentException();  
 }  
  
 AddressList addresses = new AddressList();  
  
 SAXParserFactory saxParserFactory = SAXParserFactory.*newInstance*();  
 try {  
 SAXParser saxParser = saxParserFactory.newSAXParser();  
 SaxAddressParser handler = new SaxAddressParser();  
 saxParser.parse(new File(fileName), handler);  
 ArrayList<Address> addressList = handler.getAddressList();  
  
 for (Address address : addressList) {  
 addresses.addAddress(address.getAddress());  
 }  
 } catch (ParserConfigurationException | SAXException e) {  
 throw new RuntimeException(e);  
 }  
  
 return addresses;  
 }  
}

# 2.5. Выводы

В ходе выполнения лабораторной работы мы поработали с файлами формата csv и xml. Для обработки использовались hashmap. Были использованы библиотеки, связанные с чтением файлов и hashmap.

# Разработка серверного приложения

# Это 3 практика.

# 4. Подготовка модели данных и подключение СУБД

# Это 4 практика.

# 5. Сборка и публикация результатов

Разработанный программный код собирался с помощью системы автоматизированной сборки Intellij IDEA. Результаты были выложены на Github:

<https://github.com/Gregory-hub/leti_2/tree/main/Kotlin(Java)/lab1>

<https://github.com/Tulenchi/kotlin>