**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**КАФЕДРА САПР**

**ИДЗ**

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

|  |  |  |
| --- | --- | --- |
| Студенты гр. 1301 |  | Чебесова И.Д.  Верхозина А.А. |
| Преподаватель |  | Кулагин М.В. |

Санкт-Петербург

2023

**Оглавление**

[1. Консольная программа для поиска в Википедии 3](#_Toc153500550)

[1.1. Задание 3](#_Toc153500551)

[1.2. Спецификация программы 3](#_Toc153500552)

[1.3. Описание интерфейса пользователя программы 4](#_Toc153500553)

[1.4. Текст программы 6](#_Toc153500554)

[1.5. Выводы 9](#_Toc153500555)

[2. Консольная программа для работы с CSV и XML файлами 10](#_Toc153500556)

[2.1. Задание 10](#_Toc153500557)

[2.2. Спецификация программы 10](#_Toc153500558)

[1.3. Описание интерфейса пользователя программы 12](#_Toc153500559)

[2.4. Текст программы 13](#_Toc153500560)

[2.5. Выводы 18](#_Toc153500561)

[3. Создание бота для Telegram. 20](#_Toc153500562)

[3.1. Задание 20](#_Toc153500563)

[3.2. Спецификация программы 20](#_Toc153500564)

[3.3. Описание интерфейса пользователя программы 22](#_Toc153500565)

[3.4. Текст программы 23](#_Toc153500566)

[3.5. Выводы 40](#_Toc153500567)

# Консольная программа для поиска в Википедии

# 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

В данной программе реализованы 4 класса: Main, Get\_response, Parsing и Result:

* В Main происходит ввод пользователем запроса, инициализация url строки и вызов методов других классов;
* В Get\_response метод response\_api устанавливает соединение с url строкой и возвращает данные полученные после подключения в формате String;
* В методе output класса Parsing с помощью библиотеки gson происходит выделение нужных полей исходной строки – title и pageid – и занесение последнего значения в массив для дальнейшего использования;
* В Results происходит открытие браузера с полученной добавлением нужного pageid к строке ссылкой на статью Википедии с помощью класса Desktop.

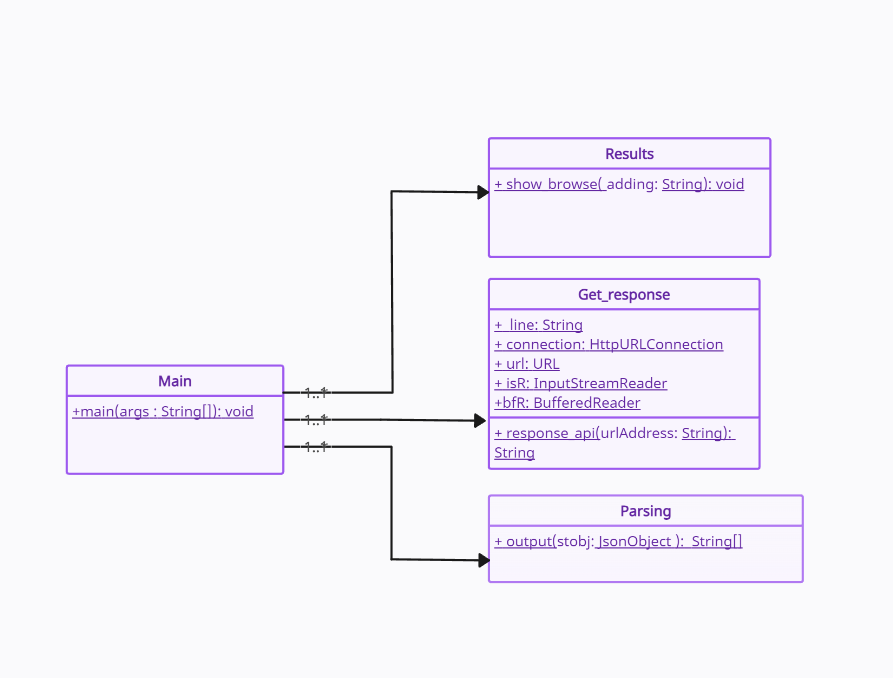
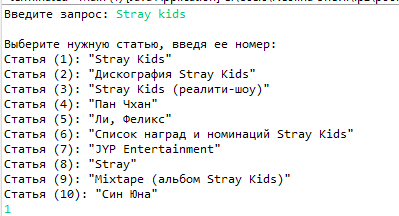
На Рисунке 1.1 отображена общая структура иерархии классов системы.

Рисунок 1.1. Диаграмма классов

# 1.3. Описание интерфейса пользователя программы

По заданию было реализовано консольное приложение, отображающее статью в браузере по выбору пользователя.

После запуска программы в консоль выводится текстовое сообщение, приглашающее пользователя ввести поисковой запрос для Википедии. Запрос может состоять как из одного слова, так и быть строкой. Далее выводятся первые 10 статей из поиска (или меньшее количество при отсутствии 10), из которых пользователь должен выбрать одну, введя ее номер (Рисунок 1.2). При корректном вводе происходит открытие браузера с выбранной статьей Википедии (Рисунок 1.3). В противном случае программа отображает текст, оповещающий о том, что запрос необходимо повторить, после чего заканчивает свою работу (Рисунок 4).

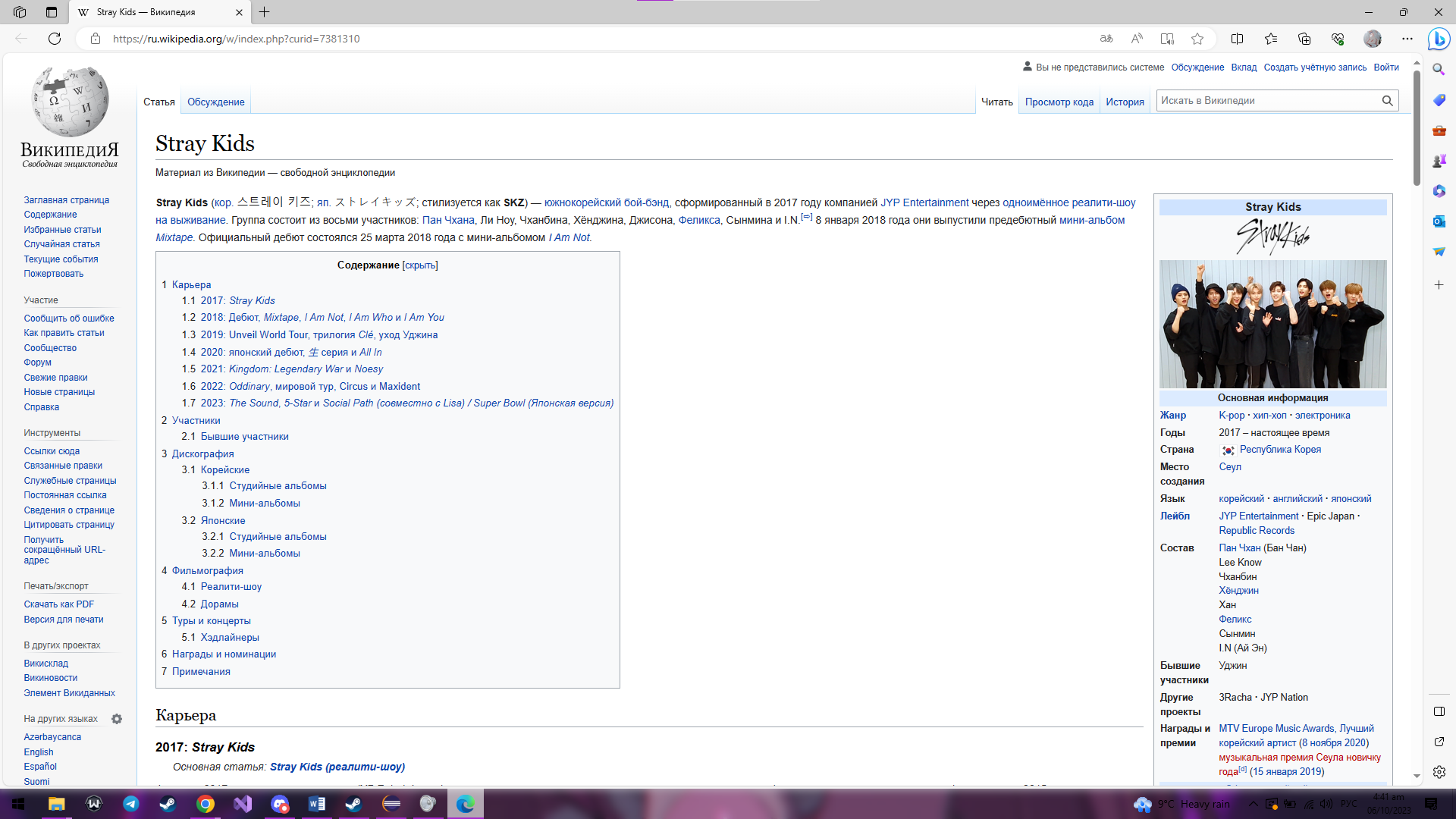
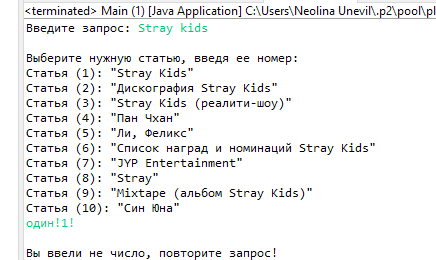
Рисунок 1.2

Рисунок 1.3

Рисунок 1.4

# 1.4. Текст программы

**Файл Main.java:**

**import** java.net.URLEncoder;

**import** java.nio.charset.StandardCharsets;

**import** java.util.InputMismatchException;

**import** java.util.Scanner;

**import** com.google.gson.JsonObject;

**import** com.google.gson.JsonParser;

**public** **class** Main {

**public** **static** void main(**String** args[])

{

//ввод запроса

**Scanner** in = **new** **Scanner**(**System**.in);

**System**.out.print("Bведите запрос: ");

**String** ask =in.nextLine();

**String** urlAddress = "https://ru.wikipedia.org/w/api.php?action=query&list=search&utf8=&format=json&srsearch="+ **URLEncoder**.encode(ask, **StandardCharsets**.UTF\_8);

**String** line="";

line= Get\_response.response\_api(urlAddress);//получение api

//распределение полученной строки по json объектам

JsonObject stobj= JsonParser.parseString(line).getAsJsonObject();

**String** pageid[]=Parsing.output(stobj);

int j=0;//индекс массива pageid

**try**

{

j =in.nextInt();

}

**catch**(**InputMismatchException** ex)

{

**System**.out.print("\nBы ввели не число, повторите запрос!\n");

**return**;

}

//этап отображения выбранной статьи в браузере

**if**(j>0&&j<pageid.length+1)

{

**String** adding = pageid[j-1];

in.close();

Results.show\_browse(adding);

}

**else**

**System**.out.print("\nBаше число выходит за границы заданного диапазона, повторите запрос!\n");

}

}

**Файл Parsing.java:**

**import** com.google.gson.JsonArray;

**import** com.google.gson.JsonObject;

**public** **class** Parsing

{

**static** **String**[] output(JsonObject stobj)

{

//создание объектов json по кодовым словам

JsonObject dataobj = stobj.getAsJsonObject("query");

JsonArray search = dataobj.getAsJsonArray("search");

**String**[] pageid =**new** **String**[search.size()];//строка для хранения pageid

**System**.out.print("\nBыберите нужную статью, введя ее номер: \n");

**for**(int i=0;i<search.size();i++)

{

JsonObject temp = search.get(i).getAsJsonObject();//получение i-того поля массива

**System**.out.printf("Статья (%d): %s\n",i+1, temp.getAsJsonPrimitive("title").toString());

pageid[i]= temp.getAsJsonPrimitive("pageid").toString();

}

**return** pageid;

}

}

**Файл Get\_response.java:**

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**public** **class** Get\_response

{

**static** **String** line;

**static** **HttpURLConnection** connection;

**static** **URL** url;

**static** **InputStreamReader** isR;

**static** **BufferedReader** bfR;

**static** **String** response\_api(**String** urlAddress)

{

**try**

{

url = **new** **URL**(urlAddress);

connection = (**HttpURLConnection**) url.openConnection();//возвращение в connection объекта URLConnection

**if** (**HttpURLConnection**.HTTP\_OK == connection.getResponseCode()) //удалось установить соединение

{

isR = **new** **InputStreamReader**(connection.getInputStream());//декодирование считанных байтов в символы во входном потоке

bfR = **new** **BufferedReader**(isR);//буферизация символов с потока для чтения далее

line = bfR.readLine();

}

**else**

{

**System**.out.printf("Fail %s", connection.getResponseCode());

}

}

**catch** (**IOException** e)

{

e.printStackTrace();

}

**return** line;

}

}

**Файл Results.java:**

**import** java.awt.Desktop;

**import** java.io.IOException;

**import** java.net.URI;

**import** java.net.URISyntaxException;

**import** java.net.URLEncoder;

**import** java.nio.charset.StandardCharsets;

**public** **class** Results

{

**static** void show\_browse(**String** adding)

{

**try**

{ //проверка на то, поддерживается ли класс и возможно ли открытие браузера

**if** (Desktop.isDesktopSupported() && Desktop.getDesktop().isSupported(Desktop.**Action**.BROWSE))

{

//возвращает экземпляр контекста браузера с указанным URI

Desktop.getDesktop().browse(**new** **URI**("https://ru.wikipedia.org/w/index.php?curid="+**URLEncoder**.encode(adding, **StandardCharsets**.UTF\_8)));

}

}

**catch** (**IOException** e)

{

**throw** **new** **RuntimeException**(e);

}

**catch** (**URISyntaxException** e)

{

**throw** **new** **RuntimeException**(e);

}

}

}

# 1.5. Выводы

В ходе данной практики были получены навыки работы с некоторыми функциями языка java для работы со ссылками. Каждый шаг выполнения представлял собой изучение возможностей новых для нас библиотек, классов и методов. Наибольший интерес в программе представляет парсинг строк с использованием библиотеки gson, так как возможно большое количество реализаций данного задания. Еще один новый для нас класс – Desktop. С помощью этого класса было реализовано открытие ссылки в браузере, среди других его функций есть также открытие приложения почты. Подводя итоги, можно сказать, что умения, полученные при написании программы, обязательно пригодятся в дальнейшем, так как они затрагивают возможности работы с интернет пространством.

# Консольная программа для работы с CSV и XML файлами

# 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

В данной программе реализованы 5 классов: Main, SAXparser, SAXhandler, CSV и HashMapping:

* В Main происходит ввод пользователем пути файла до тех пор пока не будет введена команда для выхода;
* В SAXparser метод parseзапускает обработку файла типа xml с помощью handler’а;
* В методах класса handler’а обычно описаны возможные события во время обработки – StartElement, EndElement, StartDocument, EndElement. В классе SAXhandler данной программы присутствует только метод StartElement, который передает атрибуты каждого тега на дальнейшую обработку;
* В CSV.parse происходит разделение строк из csv файла на атрибуты и передача их на дальнейшую обработку;
* HashMapping – класс, который собирает необходимую статистику с полученных данных – поиск повторов и количество этажей по всем городам. Назван так, потому что реализация сбора статистики сделана с помощью структуры данных HashMap.

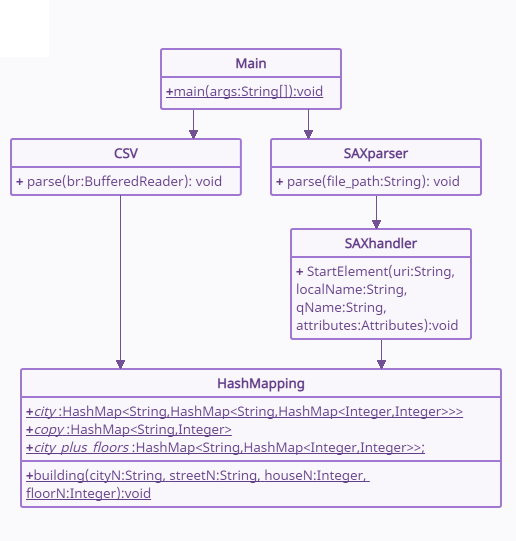
На Рисунке 2.1 отображена общая структура иерархии классов системы.

Рисунок 2.1. Диаграмма классов

# 1.3. Описание интерфейса пользователя программы

По заданию было реализовано консольное приложение. После запуска программы в консоль выводится текстовое сообщение, приглашающее пользователя ввести путь до файла или команду выхода, в данном случае - «0». После проверки файла и его обработки в консоль выводится собранная статистика: повторы (адрес, который повторился, и количество повторов), количество этажей для каждого города (название города и список возможных этажей -1, 2, 3, 4, 5 с количеством совпадений) и время обработки файла (Рис. 2.2 – Обрезанная статистика). Далее пользователю будет снова предложено ввести новый путь или ввести команду для выхода. При нахождении проблемы с файлом (неправильный тип, путь или несовпадающие с кодом программы теги внутри файла) пользователю будет выведено сообщение с просьбой повторить ввод. Примеры реакции программы на ошибки ввода пользователя – Рис. 2.3 – 2.5.

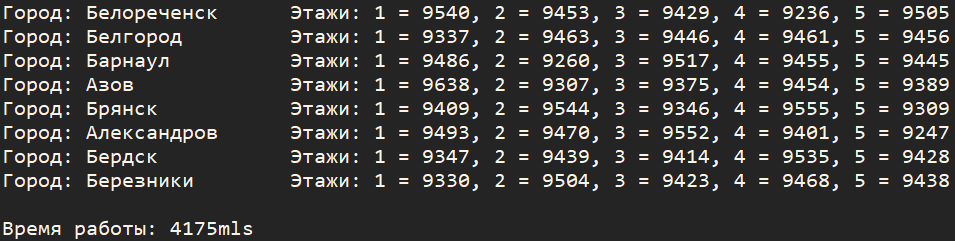
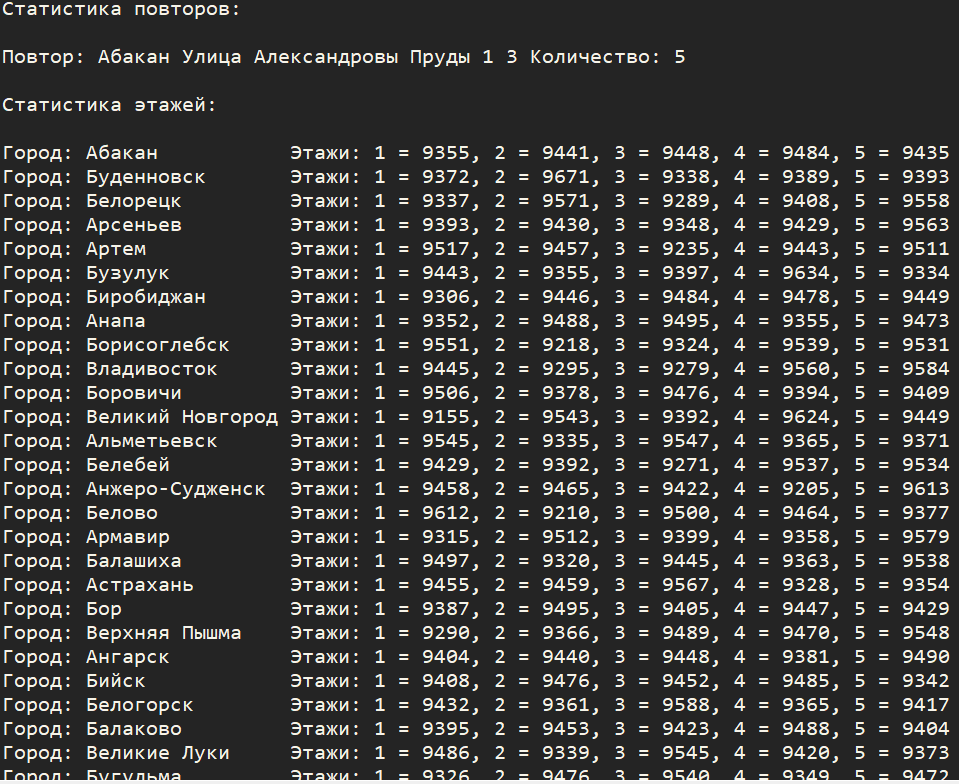
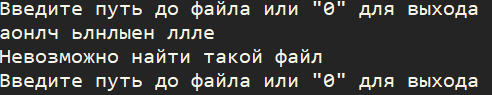
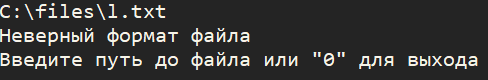


Рисунок 2.2. Обрезанная статистика

Рисунок 2.3. Неверный ввод

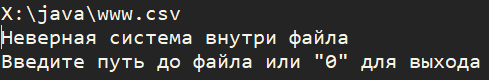
Рисунок 2.4. Неверный тип

Рисунок 2.5. Несоответствие внутреннему формату

# 2.4. Текст программы

**Файл Main.java:**

**package** lab2\_kotlin;

**import** **java.io.BufferedReader**;

**import** **java.io.FileNotFoundException**;

**import** **java.io.FileReader**;

**import** **java.util.HashMap**;

**import** **java.util.Map**;

**import** **java.util.Scanner**;

**public** **class** **Main**

{

**public** **static** **void** **main**(String args[])

{

String exit="f";

BufferedReader br=**null**;

Scanner in = **new** Scanner(System.in);

**while**(exit.equals("0")==**false**)

{

System.out.println("Введите путь до файла или \"0\" для выхода");

exit = in.nextLine();

**if**(exit.equals("0")==**false**)

{

**try**

{

br = **new** BufferedReader(**new** FileReader(exit));

}

**catch** (FileNotFoundException e)

{

System.out.println("Невозможно найти такой файл");

**continue**;

}

**if**(exit.endsWith("xml")==**false**&&exit.endsWith("csv")==**false**)

{

System.out.println("Неверный формат файла");

}

**else**

{

**long** startTime = System.currentTimeMillis();

**if**(exit.endsWith("xml")==**true**)

{

SAXparser parser = **new** SAXparser();

parser.parse(exit);

}

**if**(exit.endsWith("csv")==**true**)

{

**try**

{

CSV.parse(br);

}

**catch** (Exception e)

{

e.printStackTrace();

}

}

**long** endTime = System.currentTimeMillis();

**if**(HashMapping.city\_plus\_floors.isEmpty()==**false**)

{

System.out.println("\nСтатистика повторов:\n");

**for** (Map.Entry<String, Integer> entry : HashMapping.copy.entrySet())

{

System.out.println("Повтор: " + entry.getKey() + " Количество: " + entry.getValue());

}

System.out.println("\nСтатистика этажей:\n");

**for** (Map.Entry<String,HashMap<Integer,Integer>> entry : HashMapping.city\_plus\_floors.entrySet())

{

System.out.printf("Город: %-17s",entry.getKey());

System.out.printf("Этажи: %-10s\n",entry.getValue().toString().replace("{", "").replace("}", "").replace("=", " = "));

}

System.out.println("\nВремя работы: " + (endTime-startTime) +"mls\n");

}

}

}

HashMapping.city.clear();

HashMapping.copy.clear();

HashMapping.city\_plus\_floors.clear();

}

System.out.println("Окончание работы");

in.close();

}

}

**Файл SAXparser.java:**

**package** lab2\_kotlin;

**import** **java.io.File**;

**import** **javax.xml.parsers.SAXParser**;

**import** **javax.xml.parsers.SAXParserFactory**;

**public** **class** **SAXparser**

{

**public** **void** **parse**(String file\_path)

{

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXhandler hndl = **new** SAXhandler();

SAXParser parser = **null**;

**try**

{

parser = factory.newSAXParser();

}

**catch** (Exception e)

{

e.printStackTrace();

}

File file = **new** File(file\_path);

**try**

{

parser.parse(file, hndl);

}

**catch** (Exception e)

{

System.out.println("Неверный путь файла или файл создан неверно");

**return**;

}

}

}

**Файл SAXhandler.java:**

**package** lab2\_kotlin;

**import** **org.xml.sax.Attributes**;

**import** **org.xml.sax.SAXException**;

**import** **org.xml.sax.helpers.\***;

**public** **class** **SAXhandler** **extends** DefaultHandler

{

**@Override**

**public** **void** **startElement**(String uri, String localName, String qName, Attributes attributes) **throws** SAXException

{

**int** attributeLength = attributes.getLength();

String attr[] = **new** String[attributeLength];

**if** ("item".equals(qName))

{

**for** (**int** i = **0**; i < attributeLength; i++)

{

attr[i] = attributes.getValue(i);

}

**try**

{

HashMapping.building(attr[**0**],attr[**1**],Integer.parseInt(attr[**2**]), Integer.parseInt(attr[**3**]));

}

**catch**(Exception e)

{

System.out.println("Неверная система внутри файла");

**return**;

}

}

**else** **if**("item".equals(qName)==**false**&&"root".equals(qName)==**false**)

{

System.out.println("Неверная система внутри файла");

**return**;

}

}

}

**Файл CSV.java:**

**package** lab2\_kotlin;

**import** **java.io.BufferedReader**;

**import** **java.io.IOException**;

**public** **class** **CSV**

{

**public** **static** **int** i=**0**;

**public** **static** **void** **parse**(BufferedReader br) **throws** NumberFormatException, IOException

{

String line=**null**;

**while** ((line = br.readLine()) != **null**)

{

**if**(i!=**0**)

{

line = line.replaceAll("\"", "");

String[] attr = line.split(";");

**try**

{

HashMapping.building(attr[**0**],attr[**1**],Integer.parseInt(attr[**2**]), Integer.parseInt(attr[**3**]));

}

**catch**(Exception e)

{

System.out.println("Неверная система внутри файла");

**return**;

}

}

**else**

i++;

}

}

}

**Файл HashMapping.java:**

**package** lab2\_kotlin;

**import** **java.util.HashMap**;

**public** **class** **HashMapping**

{

**static** HashMap<String,HashMap<String,HashMap<Integer,Integer>>> city = **new** HashMap<String,HashMap<String,HashMap<Integer,Integer>>>();

**static** HashMap<String, Integer> copy = **new** HashMap<String,Integer>();

**static** HashMap<String,HashMap<Integer,Integer>> city\_plus\_floors = **new** HashMap<String,HashMap<Integer,Integer>>();

**public** **static** **void** **building**(String cityN, String streetN, Integer houseN, Integer floorN)

{

HashMap<String,HashMap<Integer,Integer>> street = **new** HashMap<String,HashMap<Integer,Integer>>();

HashMap<Integer,Integer> house = **new** HashMap<Integer,Integer>();

HashMap<Integer,Integer> each\_floor = **new** HashMap<Integer,Integer>();

**if**(city.isEmpty()==**false**&&city.get(cityN)!=**null**)//city exists:)

{

**if**(city.get(cityN).get(streetN)==**null**)//no street at the moment:(

{

street = city.get(cityN);

house.put(houseN, floorN);

street.put(streetN, house);

city.put(cityN, street);

}

**else**//this street exists:)

{

**if**(city.get(cityN).get(streetN).get(houseN)==floorN) //copy:(

{

**if**(copy.isEmpty()==**true**||copy.get(cityN+' '+streetN+' '+houseN+' '+floorN)==**null**)//first copy

{

copy.put(cityN+' '+streetN+' '+houseN+' '+floorN, **2**);

}

**else**//number'th copy

{

Integer repeat =copy.get(cityN+' '+streetN+' '+houseN+' '+floorN)+**1**;

copy.put(cityN+' '+streetN+' '+houseN+' '+floorN, repeat);

}

}

**else**//no copy:)

{

street = city.get(cityN);

house = street.get(streetN);

house.put(houseN, floorN);

street.put(streetN, house);

city.put(cityN, street);

}

}

}

**else**//city doesn't exist:(

{

house.put(houseN, floorN);

street.put(streetN, house);

city.put(cityN, street);

}

**if**(copy.get(cityN+' '+streetN+' '+houseN+' '+floorN)==**null**)//this counts if it's not a copy:)

{

**if**(city\_plus\_floors.isEmpty()==**false**&&city\_plus\_floors.get(cityN)!=**null**)//not first counter for city

{

Integer fl=city\_plus\_floors.get(cityN).get(floorN);

each\_floor = city\_plus\_floors.get(cityN);

fl++;

each\_floor.put(floorN, fl);

city\_plus\_floors.put(cityN,each\_floor);

}

**else**

{

**for**(**int** j =**1**;j<=**5**;j++)

{

each\_floor.put(j, **0**);

}

each\_floor.put(floorN, **1**);

city\_plus\_floors.put(cityN, each\_floor);

}

}

}

}

# 2.5. Выводы

В ходе данной практики были получены навыки работы с файлами типов csv и xml. Данные файлы необходимы для обмена информацией между различными приложениями и системами, из-за их многофункциональности и распространённости необходимо уметь работать с ними. В созданной программе работа с файлом csv производилась посредством строчных функций, а с файлом xml – с помощью SAX парсера (Simple API for XML). Данный парсер проходит по документу линейно сверху вниз (не читает его весь сразу), анализирует события в документе и уведомляет программу об их происхождении, после чего производятся нужные действия для их обработки. SAX был выбран из-за принципа своей работы, дающего ему преимущество в скорости при работе с большими объёмами информации.

# Создание бота для Telegram.

# 3.1. Задание

В рамках лабораторной работы требуется реализовать программу для взаимодействия с пользователями в Telegram.

Целью всей работы является создание бота, который бы позволил получить расписание занятий для любой группы. Бот должен понимать следующие команды:

near\_lesson GROUP\_NUMBER - ближайшее занятие для указанной группы;

DAY WEEK\_NUMBER GROUP\_NUMBER - расписание занятий в указанный день (monday, thuesday, ...). Неделя может быть четной, нечетной;

tommorow GROUP\_NUMBER - расписание на следующий день (если это воскресенье, то выводится расписание на понедельник, учитывая, что неделя может быть четной или нечетной);

all WEEK\_NUMBER GROUP\_NUMBER - расписание на всю неделю.

Для работы с расписанием необходимо использовать API университета:

https://digital.etu.ru/api/docs-public/

# 3.2. Спецификация программы

В данной программе реализованы 8 классов: Commands, config, Initializer, Parsing, response, Schedule\_functions, TG, DemoApplication:

* DemoApplication – класс, сгенерированный в проекте SpringBoot. Он запускает telegram бота;
* Initializer необходим для регистрации бота и запуска сессии;
* Класс Commands представляет собой enum с присвоением каждому названию команды ее пользовательского строчного вида;
* В методе output класса Parsing с помощью библиотеки gson происходит выделение нужных полей исходной строки. В возвращаемый массив заносятся массивы занятий каждого дня;
* В response метод response\_api устанавливает соединение с url строкой и возвращает данные полученные после подключения в формате String;
* Schedule\_functions содержит функции для каждой команды расписания: для нахождения ближайшего занятия, для нахождения занятий следующего дня, для вывода всего расписания, для вывода занятий определенного дня;
* Класс config содержит поля с закрытой информацией о боте и их геттеры. Значения хранятся в файле проекта со свойствами;
* Класс TG является основным для работы. В стандартном методе onUpdateReceived происходит проверка введенных команд и текста, а также обработка ответов.

На Рисунке 3.2 отображена общая структура иерархии классов системы.

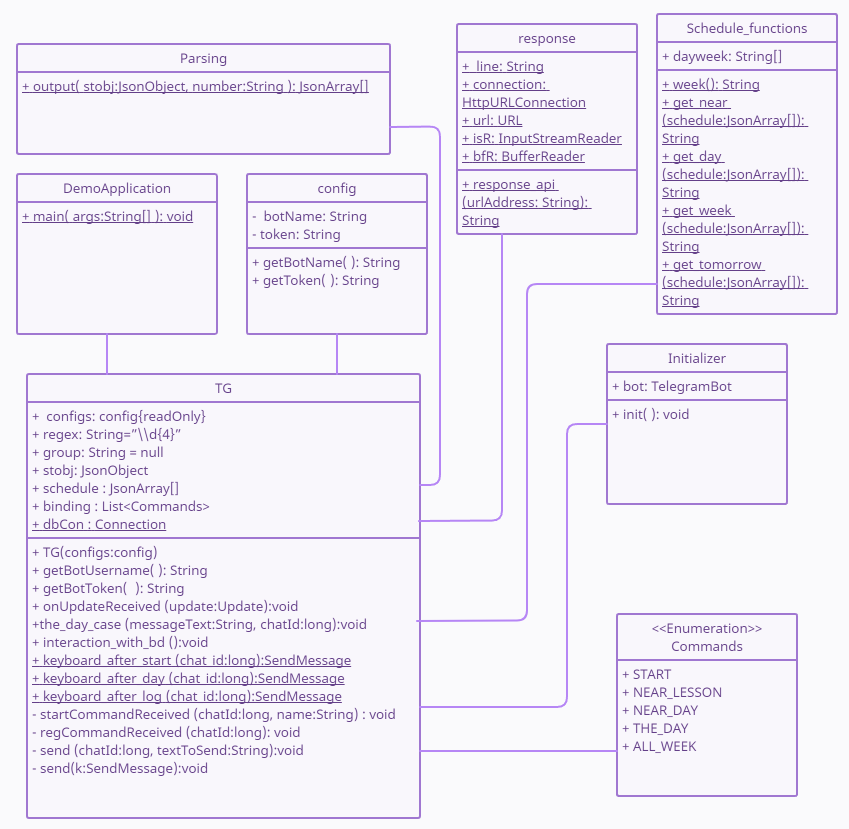


Рис. 3.2

# 3.3. Описание интерфейса пользователя программы

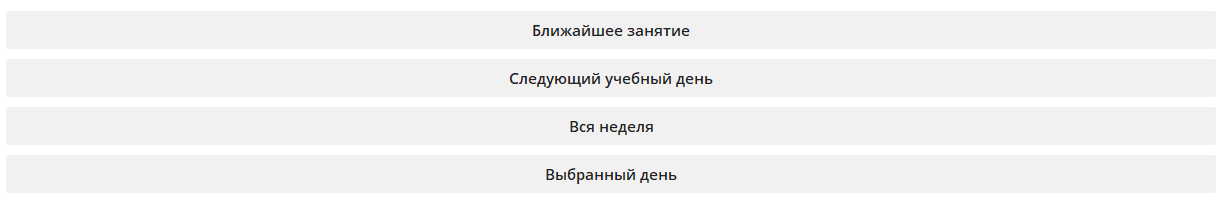
 Отображение программы происходит в программе telegram. Все выводимые сообщение выводятся стандартными шрифтами приложения. Для отображения информации и удобства пользователя реализованы reply кнопки, которые выводятся при необходимости пользовательского ввода под текстовым окном и при вводе текста скрываются. Всего реализовано 2 такие клавиатуры: для всех функций расписания и для вывода дней недели при выборе функции расписания для определенного дня.

Рис. 3.3.1

Рис. 3.3.2

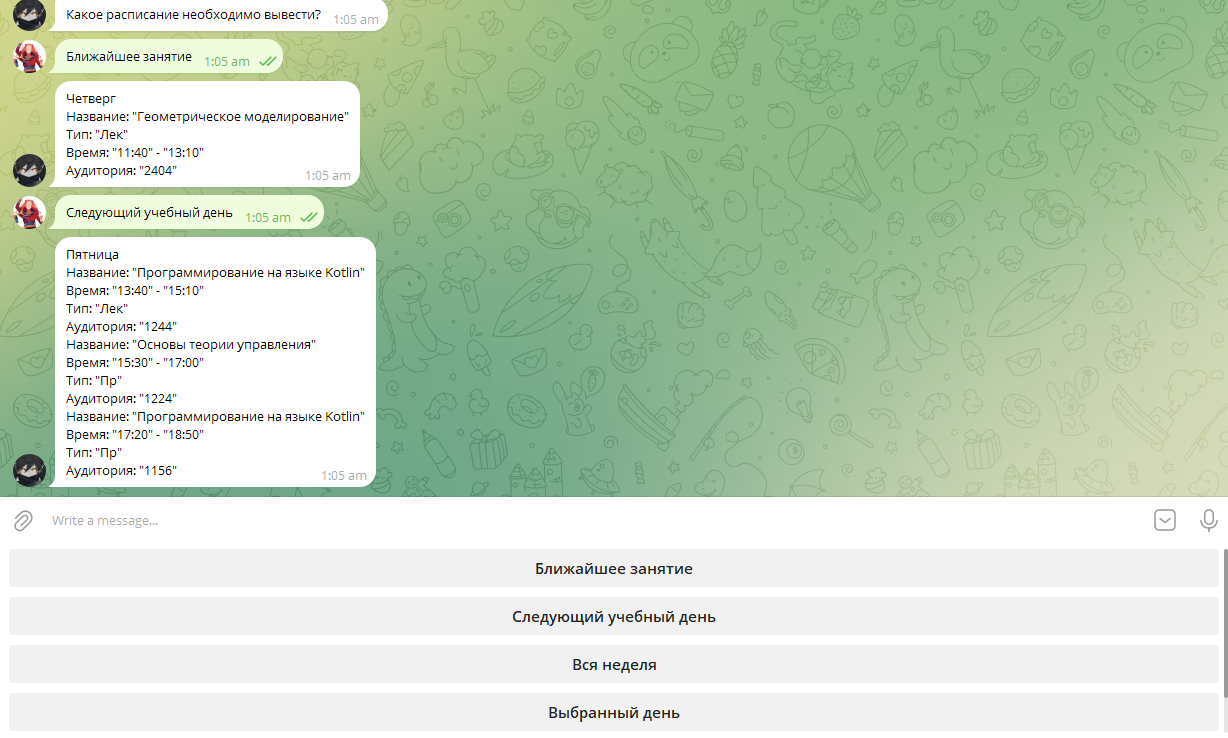
Вывод происходит в виде обычных текстовых сообщений.

Рис. 3.3.3

# 3.4. Текст программы

**Файл Commands.java:**

**package** com.example.demo;

**public** **enum** Commands {

START("/start"),

NEAR\_LESSON("Ближайшее занятие"),

NEAR\_DAY("Следующий учебный день"),

ALL\_WEEK("Вся неделя"),

THE\_DAY("Выбранный день");

**public** **final** String cmd;

Commands(String cmd)

{

**this**.cmd=cmd;

}

**@Override**

**public** String **toString**()

{

**return** cmd;

}

**public** **boolean** **equals**(String cmd)

{

**return** **this**.toString().equals(cmd);

}

}

**Файл config.java:**

**package** com.example.demo;

**import** **lombok.Data**;

**import** **org.springframework.beans.factory.annotation.Value**;

**import** **org.springframework.context.annotation.Configuration**;

**import** **org.springframework.context.annotation.PropertySource**;

**@Configuration**

**@Data**

**@PropertySource**("application.properties")

**public** **class** **config** {

**@Value**("${bot.name}")

**private** String botName;

**@Value**("${bot.token}")

**private** String token;

**public** String **getBotName**() {

**return** botName;

}

**public** String **getToken**() {

**return** token;

}

}

**Файл DemoApplication.java:**

**package** com.example.demo;

**import** **org.springframework.boot.SpringApplication**;

**import** **org.springframework.boot.autoconfigure.SpringBootApplication**;

**@SpringBootApplication**

**public** **class** **DemoApplication** {

**public** **static** **void** **main**(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**Файл Initializer.java:**

**package** com.example.demo;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.context.event.ContextRefreshedEvent**;

**import** **org.springframework.context.event.EventListener**;

**import** **org.springframework.stereotype.Component**;

**import** **org.telegram.telegrambots.meta.TelegramBotsApi**;

**import** **org.telegram.telegrambots.meta.exceptions.TelegramApiException**;

**import** **org.telegram.telegrambots.meta.generics.LongPollingBot**;

**import** **org.telegram.telegrambots.meta.generics.TelegramBot**;

**import** **org.telegram.telegrambots.updatesreceivers.DefaultBotSession**;

**@Component**

**public** **class** **Initializer** {

**@Autowired**

TelegramBot bot;

**@EventListener**({ContextRefreshedEvent.class})

**public** **void** **init**() **throws** TelegramApiException {

TelegramBotsApi telegramBotsApi = **new** TelegramBotsApi(DefaultBotSession.class);

**try** {

telegramBotsApi.registerBot((LongPollingBot) bot);

}

**catch** (TelegramApiException e)

{

}

}

}

**Файл Parsing.java:**

**package** com.example.demo;

**import** **com.google.gson.JsonArray**;

**import** **com.google.gson.JsonObject**;

**public** **class** **Parsing** {

**static** JsonArray[] **output**(JsonObject stobj,String number)

{

//создание объектов json по кодовым словам

JsonObject group = stobj.getAsJsonObject(number);

JsonObject days = group.getAsJsonObject("days");

JsonObject[] week = **new** JsonObject[**7**];

JsonArray[] lessons = **new** JsonArray[**7**];

**for**(Integer i=**0**;i<**7**;i++)

{

week[i]=days.getAsJsonObject(i.toString());

lessons[i]=week[i].getAsJsonArray("lessons");

}

**return** lessons;

}

}

**Файл rsponse.java:**

**package** com.example.demo;

**import** **java.io.BufferedReader**;

**import** **java.io.IOException**;

**import** **java.io.InputStreamReader**;

**import** **java.net.HttpURLConnection**;

**import** **java.net.URL**;

**public** **class** **rsponse** {

**static** String line;

**static** HttpURLConnection connection;

**static** URL url;

**static** InputStreamReader isR;

**static** BufferedReader bfR;

**static** String **response\_api**(String urlAddress)

{

**try**

{

url = **new** URL(urlAddress);

connection = (HttpURLConnection) url.openConnection();//возвращение в connection объекта URLConnection

**if** (HttpURLConnection.HTTP\_OK == connection.getResponseCode()) //удалось установить соединение

{

isR = **new** InputStreamReader(connection.getInputStream());//декодирование считанных байтов в символы во входном потоке

bfR = **new** BufferedReader(isR);//буферизация символов с потока для чтения далее

line = bfR.readLine();

}

**else**

{

System.out.printf("Fail %s", connection.getResponseCode());

}

}

**catch** (IOException e)

{

e.printStackTrace();

}

**return** line;

}

}

**Файл Schedule\_functions.java:**

**package** com.example.demo;

**import** **java.text.ParseException**;

**import** **java.text.SimpleDateFormat**;

**import** **java.time.LocalDate**;

**import** **java.time.LocalTime**;

**import** **java.util.Date**;

**import** **com.google.gson.JsonArray**;

**import** **com.google.gson.JsonObject**;

**public** **class** **Schedule\_functions** {

**static** String[] dayweek = {" Понедельник "," Вторник "," Среда "," Четверг "," Пятница "," Суббота "," Воскресенье "};

**public** **static** String **week**()

{

Date start = **null**;

**try** {

start = **new** SimpleDateFormat("dd/MM/yyyy").parse("01/09/2023");

} **catch** (ParseException exc) {

exc.printStackTrace();

}

**long** delay = System.currentTimeMillis() - start.getTime();

**long** week = **1000** \* **60** \* **60** \* **24** \* **7**;

delay %= week \* **2**;

**if** (delay <= week)

**return** "\"1\"";

**else**

**return** "\"2\"";

}

**public** **static** String **get\_near**(JsonArray[] schedule)

{

LocalDate date = LocalDate.now();

**int** hours= LocalTime.now().getHour();

**int** minutes =LocalTime.now().getMinute();

**int** dow=date.getDayOfWeek().getValue()-**1**;

String quanity = week();

JsonArray needed\_day=schedule[dow];

JsonObject lesson = **new** JsonObject();

**boolean** found = **false**;

**for**(**int** i=**0**;i<needed\_day.size();i++)

{

lesson = needed\_day.get(i).getAsJsonObject();

String endTime = lesson.getAsJsonPrimitive("end\_time").toString();

endTime=endTime.replace("\"", "");

String[] parts = endTime.split(":");

**int** lhour=Integer.valueOf(parts[**0**]);

**int** lmin=Integer.valueOf(parts[**1**]);

**if**(hours<lhour||(hours==lhour&&minutes<=lmin)&&((quanity==lesson.getAsJsonPrimitive("week").toString())||(lesson.getAsJsonPrimitive("week").toString().equals("\"3\""))))

{

found=**true**;

**break**;

}

}

**while**(found==**false**)

{ dow=(dow+**1**)%**7**;

**if**(dow==**0**)

{

**if**(quanity.equals("\"1\""))

quanity="\"2\"";

**else**

quanity="\"1\"";

}

needed\_day=schedule[dow];

**if**(needed\_day.isEmpty()==**false**)

{

**for**(**int** i=**0**;i<needed\_day.size();i++)

{

lesson = needed\_day.get(i).getAsJsonObject();

**if**((quanity.equals(lesson.getAsJsonPrimitive("week").toString()))||(lesson. getAsJsonPrimitive("week").toString().equals("3")==**true**)) { found=**true**;

**break**; } }

} }

String fin;

fin=dayweek[dow] +"\nНазвание: "+ lesson.getAsJsonPrimitive("name").toString()+"\n"+"Тип: "+lesson.getAsJsonPrimitive("subjectType").toString()+"\nВремя: "+lesson.getAsJsonPrimitive("start\_time").toString() + " - " + lesson.getAsJsonPrimitive("end\_time").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("room").toString().equals("\"\"")==**false**)

fin+="Аудитория: "+lesson.getAsJsonPrimitive("room").toString();

**return** fin;

}

**public** **static** String **get\_day**(JsonArray[] schedule,**int** day)

{

JsonArray needed\_day=schedule[day];

**if**(needed\_day.isEmpty()==**false**)

{ String fin=dayweek[day]+"\n";

**for**(**int** i=**0**;i<needed\_day.size();i++)

{

JsonObject lesson = needed\_day.get(i).getAsJsonObject();

fin+="Название: "+lesson.getAsJsonPrimitive("name").toString()+"\n";

fin+="Тип: "+lesson.getAsJsonPrimitive("subjectType").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("week").toString().equals("\"3\"")==**false**)

fin+="Четность недели : "+lesson.getAsJsonPrimitive("week").toString()+"\n";

fin+="Время: "+lesson.getAsJsonPrimitive("start\_time").toString() + " - " + lesson.getAsJsonPrimitive("end\_time").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("room").toString().equals("\"\"")==**false**)

fin+="Аудитория: "+lesson.getAsJsonPrimitive("room").toString()+"\n";

}

**return** fin;

}

**else**

**return** dayweek[day]+"\n"+"Нет занятий в этот день!";

}

**public** **static** String **get\_tomorrow**(JsonArray[] schedule)

{

LocalDate date = LocalDate.now();

**int** dow=date.getDayOfWeek().getValue()-**1**;

String quanity = week();

dow=dow+**1**;

JsonArray needed\_day=schedule[dow];

**while**(needed\_day.isEmpty()==**true**)

{

dow=(dow+**1**)%**7**;

**if**(dow==**0**)

{

**if**(quanity.equals("\"1\""))

quanity="\"2\"";

**else**

quanity="\"1\"";

}

needed\_day=schedule[dow];

}

String fin=dayweek[dow]+"\n";

**for**(**int** i=**0**;i<needed\_day.size();i++)

{

JsonObject lesson = needed\_day.get(i).getAsJsonObject();

**if**(lesson.getAsJsonPrimitive("week").toString().equals("\"3\"")||quanity.equals(lesson.getAsJsonPrimitive("week").toString()))

{

fin+="Название: "+lesson.getAsJsonPrimitive("name").toString()+"\n";

fin+="Время: "+lesson.getAsJsonPrimitive("start\_time").toString() + " - " + lesson.getAsJsonPrimitive("end\_time").toString()+"\n";

fin+="Тип: "+lesson.getAsJsonPrimitive("subjectType").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("room").toString().equals("\"\"")==**false**)

fin+="Аудитория: "+lesson.getAsJsonPrimitive("room").toString()+"\n";

}

}

**return** fin;

}

**public** **static** String **get\_week**(JsonArray[] schedule)

{

String fin = **new** String();

**for**(**int** i=**0**;i<**7**;i++)

{

JsonArray needed\_day=schedule[i];

**if**(needed\_day.isEmpty()==**false**)

fin+=(dayweek[i]);

**for**(**int** j=**0**;j<needed\_day.size();j++)

{

**if**(needed\_day.isEmpty()==**false**)

{

JsonObject lesson = needed\_day.get(j).getAsJsonObject();

fin+="Название: "+lesson.getAsJsonPrimitive("name").toString()+"\n";

fin+="Тип: "+lesson.getAsJsonPrimitive("subjectType").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("week").toString().equals("\"3\"")==**false**)

fin+="Четность недели : "+lesson.getAsJsonPrimitive("week").toString()+"\n";

fin+="Время: "+lesson.getAsJsonPrimitive("start\_time").toString() + " - " + lesson.getAsJsonPrimitive("end\_time").toString()+"\n";

**if**(lesson.getAsJsonPrimitive("room").toString().equals("\"\"")==**false**)

fin+="Аудитория: "+lesson.getAsJsonPrimitive("room").toString()+"\n";

}

}

}

**return** fin;

}

}

**Файл TG.java:**

**package** com.example.demo;

**import** **java.net.URLEncoder**;

**import** **java.sql.Connection**;

**import** **java.sql.DriverManager**;

**import** **java.sql.PreparedStatement**;

**import** **java.sql.ResultSet**;

**import** **java.sql.SQLException**;

**import** **java.sql.Statement**;

**import** **java.nio.charset.StandardCharsets**;

**import** **java.util.ArrayList**;

**import** **java.util.EnumSet**;

**import** **java.util.HashMap**;

**import** **java.util.List**;

**import** **org.springframework.stereotype.Component**;

**import** **org.telegram.telegrambots.bots.TelegramLongPollingBot**;

**import** **org.telegram.telegrambots.meta.api.methods.send.SendMessage**;

**import** **org.telegram.telegrambots.meta.api.objects.Update**;

**import** **org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup**;

**import** **org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow**;

**import** **org.telegram.telegrambots.meta.exceptions.TelegramApiException**;

**import** **com.google.gson.JsonArray**;

**import** **com.google.gson.JsonObject**;

**import** **com.google.gson.JsonParser**;

**@Component**

**public** **class** **TG** **extends** TelegramLongPollingBot

{

**final** config configs;

String regex ="\\d{4}";

String group=**null**;

String login=**null**;

JsonObject stobj;

JsonArray[] schedule = **new** JsonArray[**7**];

List<Commands> binding = **new** ArrayList<Commands>();

**static** Connection dbCon;

**@SuppressWarnings**("deprecation")

**public** **TG**(config configs)

{

**this**.configs=configs;

}

**@Override**

**public** String **getBotUsername**()

{

**return** configs.getBotName();

}

**@Override**

**public** String **getBotToken**()

{

**return** configs.getToken();

}

**public** **static** Connection **getConnection**()

**throws** SQLException, ClassNotFoundException

{ String connectionString = "jdbc:mysql://localhost:3306/Storage";

Class.forName("com.mysql.cj.jdbc.Driver");

Connection dbConnection = DriverManager.getConnection(connectionString, "root", "SolThe1stOfficer");

**return** dbConnection;

}

**@Override**

**public** **void** **onUpdateReceived**(Update update)

{

**if** (update.hasMessage() && update.getMessage().hasText())

{

String messageText = update.getMessage().getText();

**long** chatId = update.getMessage().getChatId();

**if**(!binding.contains(Commands.LOGGED))//проверка на логин

{

**if**(Commands.START.equals(messageText))

{

startCommandReceived(chatId,update.getMessage().getChat().getFirstName());

binding.add(Commands.START);

}

**else** **if**(Commands.REGISTRATION.equals(messageText)&&binding.contains(Commands.START))

{

regCommandReceived(chatId);

binding.clear();

binding.add(Commands.REGISTRATION);

}

**else** **if**(Commands.LOGIN.equals(messageText)&&binding.contains(Commands.START))

{

regCommandReceived(chatId);

binding.clear();

binding.add(Commands.LOGIN);

}

**else** **if**(binding.contains(Commands.REGISTRATION)||binding.contains(Commands.LOGIN))

{

login=messageText;

interaction\_with\_bd();

**if**(dbCon!=**null**)

{

Statement st=**null**;

ResultSet rs=**null**;

String Sql = "select ugroup from users\_info where login='"+login+"'";

**try**

{

st = dbCon.createStatement();

rs = st.executeQuery(Sql);

**while**(rs.next())

group=rs.getString("ugroup");

}

**catch** (SQLException e)

{

e.printStackTrace();

}

**if**(!binding.contains(Commands.REGISTRATION))

{**if**(group!=**null**)

{

binding.clear();

binding.add(Commands.LOGGED);

send(chatId,"Вы успешно авторизованы!");

String urlAddress = "https://digital.etu.ru/api/mobile/schedule?weekDay=&subjectType=&joinWeeks=true&year=2023&groupNumber="+ URLEncoder.encode(group, StandardCharsets.UTF\_8);

String line="";

line= rsponse.response\_api(urlAddress);

stobj= JsonParser.parseString(line).getAsJsonObject();

schedule = Parsing.output(stobj,group);

send(keyboard\_after\_log(chatId));

}

**else** **if**(group==**null**)

{

send(chatId,"Такого логина нет в базе данных! Проверьте ввод и повторите попытку или выберите 'Регистрация'");

binding.add(Commands.START);

send(keyboard\_after\_start(chatId));

}}

**else**

{

**if**(group==**null**)

{

binding.clear();

binding.add(Commands.LOGIN\_INPUTED);

send(chatId,"Введите номер группы.");

}

**else** **if**(group!=**null**)

{

send(chatId,"Такой пользователь уже существует!Повторите ввод логина или выберите 'Вход'");

binding.add(Commands.START);

send(keyboard\_after\_start(chatId));

group=**null**;

}

}

}

**else**

**send**(chatId,"Не удалось установить соединение с базой данных!");

}

**else** **if**(binding.contains(Commands.LOGIN\_INPUTED))

{

**if**(group==**null**)

group=messageText;

String urlAddress = "https://digital.etu.ru/api/mobile/schedule?weekDay=&subjectType=&joinWeeks=true&year=2023&groupNumber="+ URLEncoder.encode(group, StandardCharsets.UTF\_8);

String line="";

line= rsponse.response\_api(urlAddress);

**if**(group.matches(regex)&&!line.equals("{}"))

{

**if**(binding.contains(Commands.LOGIN\_INPUTED))

{

binding.clear();

binding.add(Commands.LOGGED);

String sql = "insert into users\_info values('"+login+"','"+group+"')";

**try**

{

PreparedStatement pst = dbCon.prepareStatement(sql);

pst.executeUpdate();

}

**catch** (SQLException e)

{

e.printStackTrace();

}

send(chatId,"Вы успешно зарегистрированы!");

}

stobj= JsonParser.parseString(line).getAsJsonObject();

schedule = Parsing.output(stobj,group);

send(keyboard\_after\_log(chatId));

}

**else**

{

send(chatId,"Такой группы в расписании не существует! Повторите ввод!");

group=**null**;

}

}

}

**else**

{

**if**(!binding.contains(Commands.THE\_DAY))

{

**if**(Commands.NEAR\_DAY.equals(messageText))

{

String mes=Schedule\_functions.get\_tomorrow(schedule);

send(chatId,mes);

}

**else** **if**(Commands.ALL\_WEEK.equals(messageText))

{

String mes=Schedule\_functions.get\_week(schedule);

send(chatId,mes);

}

**else** **if**(Commands.NEAR\_LESSON.equals(messageText))

{

String mes=Schedule\_functions.get\_near(schedule);

send(chatId,mes);

}

**else** **if**(Commands.THE\_DAY.equals(messageText))

{

send(keyboard\_after\_day(chatId));

binding.add(Commands.THE\_DAY);

}

**else** **if**(Commands.EXIT.equals(messageText))

{

send(chatId,"Вы вышли из своего аккаунта!");

startCommandReceived(chatId,update.getMessage().getChat().getFirstName());

binding.clear();

binding.add(Commands.START);

}

}

**else**

{

the\_day\_case(messageText,chatId);

binding.remove(Commands.THE\_DAY);

}

}

}

}

**public** **void** **the\_day\_case**(String messageText,**long** chatId)

{

**int** wday=**0**;

**switch**(messageText)

{

**case** "Понедельник":

{

wday=**0**;

**break**;

}

**case** "Вторник":

{

wday=**1**;

**break**;

}

**case** "Среда":

{

wday=**2**;

**break**;

}

**case** "Четверг":

{

wday=**3**;

**break**;

}

**case** "Пятница":

{

wday=**4**;

**break**;

}

**case** "Суббота":

{

wday=**5**;

**break**;

}

}

String mes=Schedule\_functions.get\_day(schedule,wday);

send(chatId,mes);

send(keyboard\_after\_log(chatId));

}

**public** **static** **void** **interaction\_with\_bd**()

{

**try**

{

dbCon = getConnection();

}

**catch** (SQLException e)

{

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

}

**public** **static** SendMessage **keyboard\_after\_start** (**long** chat\_id)

{

SendMessage message = **new** SendMessage();

message.setChatId(chat\_id);

message.setText("Для продолжения работы войдите или зарегистрируйтесь");

ReplyKeyboardMarkup markupInline = **new** ReplyKeyboardMarkup();

markupInline.setSelective(**true**);

markupInline.setResizeKeyboard(**true**);

markupInline.setOneTimeKeyboard(**true**);

List<KeyboardRow> keyboard = **new** ArrayList<>();

KeyboardRow keyboardFirstRow = **new** KeyboardRow();

keyboardFirstRow.add("Вход");

keyboardFirstRow.add("Регистрация");

keyboard.add(keyboardFirstRow);

markupInline.setKeyboard(keyboard);

message.setReplyMarkup(markupInline);

**return** message;

}

**public** **static** SendMessage **keyboard\_after\_day** (**long** chat\_id)

{

SendMessage message = **new** SendMessage();

message.setChatId(chat\_id);

message.setText("Какое расписание необходимо вывести?");

ReplyKeyboardMarkup markupR = **new** ReplyKeyboardMarkup();

markupR.setSelective(**true**);

markupR.setResizeKeyboard(**true**);

markupR.setOneTimeKeyboard(**true**);

List<KeyboardRow> keyboard = **new** ArrayList<>();

KeyboardRow k1Row = **new** KeyboardRow();

k1Row.add("Понедельник");

KeyboardRow k2Row = **new** KeyboardRow();

k2Row.add("Вторник");

KeyboardRow k3Row = **new** KeyboardRow();

k3Row.add("Среда");

KeyboardRow k4Row = **new** KeyboardRow();

k4Row.add("Четверг");

KeyboardRow k5Row = **new** KeyboardRow();

k5Row.add("Пятница");

KeyboardRow k6Row = **new** KeyboardRow();

k6Row.add("Суббота");

keyboard.add(k1Row);

keyboard.add(k2Row);

keyboard.add(k3Row);

keyboard.add(k4Row);

keyboard.add(k5Row);

keyboard.add(k6Row);

markupR.setKeyboard(keyboard);

message.setReplyMarkup(markupR);

**return** message;

}

**public** **static** SendMessage **keyboard\_after\_log** (**long** chat\_id)

{

SendMessage message = **new** SendMessage();

message.setChatId(chat\_id);

message.setText("Какое расписание необходимо вывести?");

ReplyKeyboardMarkup markupR = **new** ReplyKeyboardMarkup();

markupR.setSelective(**true**);

markupR.setResizeKeyboard(**true**);

markupR.setOneTimeKeyboard(**false**);

List<KeyboardRow> keyboard = **new** ArrayList<>();

KeyboardRow k1Row = **new** KeyboardRow();

k1Row.add("Ближайшее занятие");

KeyboardRow k2Row = **new** KeyboardRow();

k2Row.add("Следующий учебный день");

KeyboardRow k3Row = **new** KeyboardRow();

k3Row.add("Вся неделя");

KeyboardRow k4Row = **new** KeyboardRow();

k4Row.add("Выбранный день");

KeyboardRow k5Row = **new** KeyboardRow();

k5Row.add("Выход");

keyboard.add(k1Row);

keyboard.add(k2Row);

keyboard.add(k3Row);

keyboard.add(k4Row);

keyboard.add(k5Row);

markupR.setKeyboard(keyboard);

message.setReplyMarkup(markupR);

**return** message;

}

**private** **void** **startCommandReceived**(**long** chatId, String name)

{

String answer = "Привет, " + name + ", это Бот для получения расписания!\nДля того, чтобы начать, Вам необходимо авторизоваться.";

send(chatId, answer);

send(keyboard\_after\_start(chatId));

}

**private** **void** **regCommandReceived**(**long** chatId) {

String answer = "Введите Ваш логин.";

send(chatId, answer);

}

**private** **void** **send**(**long** chatId, String textToSend)

{

SendMessage message = **new** SendMessage();

message.setChatId(String.valueOf(chatId));

message.setText(textToSend);

**try** {

execute(message);

} **catch** (TelegramApiException e) {

e.printStackTrace();

}

}

**private** **void** **send**(SendMessage k)

{

**try** {

execute(k);

} **catch** (TelegramApiException e) {

e.printStackTrace();

}

}

}

# 3.5. Выводы

В ходе данной практики были получены навыки работы создания бота telegram и использования сторонних api для его реализации. Был использован генератор проектов SpringBoot, в файлах которого уже присутствуют все необходимые связи. Также для реализации вывода расписания были использованы функции из предыдущих лабораторных работ, которые были успешно интегрированы в выполнение нового функционала.