**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Создание бота для Telegram»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3354 |  | Филимонова Е.С. |
| Преподаватель |  | Кулагин М.В. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |  |
| --- | --- | --- |
| Студентка Филимонова Е.С. | | |
| Группа 3354 | | |
| Тема работы: «Создание бота для Telegram» | | |
| Исходные данные:  Язык программирования: Java версия 8 update 351. ПО для разработки IntelliJ IDEA Community Edition 2024.2.1 | | |
| Содержание пояснительной записки:  Аннотация, содержание, введение, первый раздел, заключение, список использованных источников, приложение. | | |
|  | | |
| Дата выдачи задания: 04.12.2024 | | |
| Дата сдачи реферата: 12.12.2024 | | |
| Дата защиты реферата: 18.12.2024 | | |
| Студентка |  | Филимонова Е.С. |
| Преподаватель |  | Кулагин М.В. |

**АННОТАЦИЯ**

В рамках курсового проекта разработан Telegram-бот для предоставления информации о погоде. Основная цель проекта — создание удобного инструмента, который позволяет пользователю получать текущую погоду, прогноз на завтра и скорость ветра для выбранного города.

Использованы API OpenWeatherMap для получения данных о погоде. Применение библиотеки Telegram Bots для взаимодействия с Telegram API. Реализация пользовательского интерфейса через клавиатуры Telegram. Обработка данных с использованием JSON для парсинга ответа API.

Бот успешно предоставляет информацию о погоде, включая описание, температуру, советы по одежде и данные о ветре. Реализован функционал выбора популярных городов и заданных команд. Достигнута стабильная работа бота с обработкой ошибок и персонализацией запросов для каждого пользователя.

**SUMMARY**

As part of the course project, a Telegram bot has been developed to provide weather information. The main goal of the project is to create a convenient tool that allows the user to get the current weather, forecast for tomorrow and wind speed for the selected city.

The OpenWeatherMap API is used to get weather data. Using the Telegram Bots library to interact with the Telegram API. Implementation of the user interface via Telegram keyboards. Processing data using JSON to parse the API response.

The bot successfully provides weather information, including description, temperature, clothing tips, and wind data. The functionality of selecting popular cities and specified teams has been implemented. The stable operation of the bot with error handling and personalization of requests for each user has been achieved.

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Аннотация | 3 |
|  | Содержание | 4 |
|  | Введение | 5 |
| 1. | ОПИСАНИЕ TELEGRAM - БОТА | 6 |
| 1.1. | Диаграмма вариантов использования | 6 |
| 1.2. | Диаграмма классов объектной модели предметной области | 7 |
| 1.3. | Спецификация классов | 9 |
| 1.4. | Код классов объектной модели | 11 |
| 1.5. | Интерфейс программы | 16 |
|  | Заключение | 21 |
|  | Список использованных источников | 22 |
|  | Приложение А. Листинг | 23 |

**ВВЕДЕНИЕ**

**Цель работы:**

Разработка Telegram-бота, предоставляющего информацию о погоде в удобной и интерактивной форме.

**Основные задачи:**

1. Интеграция с API OpenWeatherMap для получения актуальных данных о погоде.
2. Реализация взаимодействия с пользователями через Telegram, включая обработку текстовых сообщений и предоставление предустановленных клавиатур.
3. Разработка функций для получения прогноза на завтра, скорости ветра и текущей погоды.
4. Обеспечение корректной обработки пользовательских ошибок и нераспознанных запросов.

**Методы решения задач:**

* Использование Java для разработки программного обеспечения.
* Применение Telegram Bots API для создания пользовательских интерфейсов и обработки запросов.
* Работа с HTTP-запросами для взаимодействия с погодным API и JSON для обработки данных.
* Обеспечение стабильной работы бота через логирование, обработку исключений и поддержку диалогового состояния пользователя.
  1. **ОПИСАНИЕ TELEGRAM - БОТА**

Для реализации программы Telegram-бота для взаимодействия с пользователями выберем тему бота, а также определим варианты использования. Актуальной и необходимой практически каждый день темой бота является прогноз погоды, поэтому остановимся на этой теме.

* 1. **Диаграмма вариантов использования**

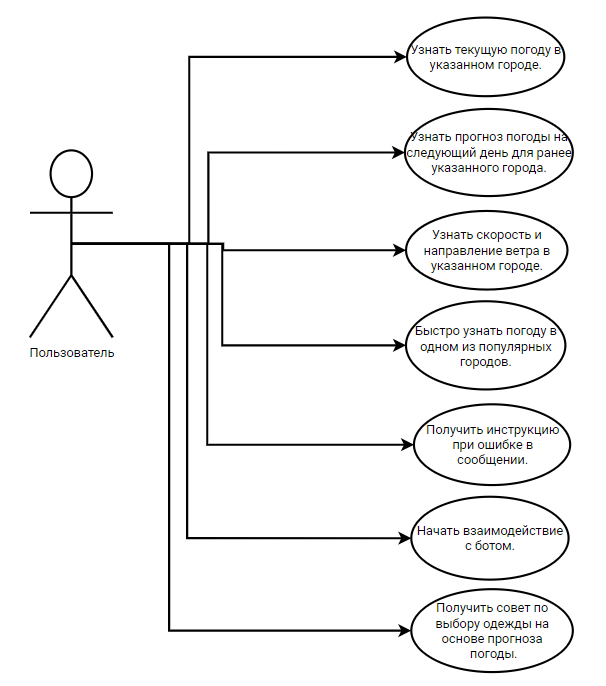


Рис. 1. Диаграмма вариантов использования системы (use cases)

* 1. **Диаграмма классов объектной модели предметной области.**

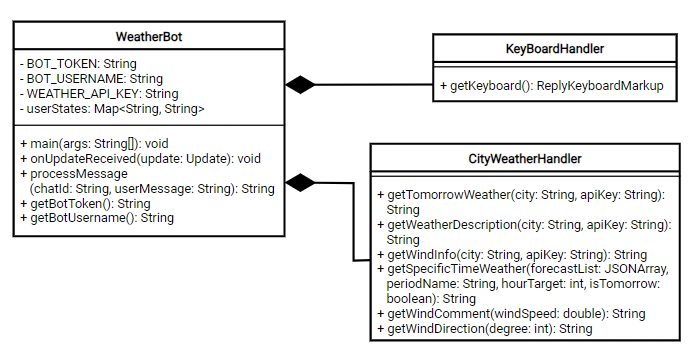


Рис. 2. Диаграмма классов

Описание классов программы:

1. **WeatherBot:**

Этот класс является основным для работы бота. Он наследует TelegramLongPollingBot из Telegram API, чтобы обрабатывать сообщения пользователей.

Атрибуты:

BOT\_TOKEN: String — токен для доступа к Telegram Bot API.

BOT\_USERNAME: String — имя пользователя бота.

WEATHER\_API\_KEY: String — API-ключ для доступа к OpenWeatherMap.

userStates: Map<String, String> — хранилище текущего состояния пользователей (связывает ID чата с текущим выбранным городом).

Методы:

main(args: String[]): void — запускает бот и регистрирует его с помощью TelegramBotsApi.

onUpdateReceived(update: Update): void — обрабатывает входящие сообщения и вызывает соответствующие методы.

processMessage(chatId: String, userMessage: String): String — определяет ответ на пользовательское сообщение. Сохраняет выбранный город и вызывает методы из CityWeatherHandler.

getBotUsername(): String — возвращает имя бота.

getBotToken(): String — возвращает токен бота.

1. **KeyboardHandler**

Этот класс отвечает за создание пользовательского интерфейса (клавиатуры).

Методы:

getKeyboard(): ReplyKeyboardMarkup — возвращает клавиатуру с предустановленными кнопками (например, города, прогноз на завтра, скорость ветра).

Кнопки разделены на ряды:

Первый ряд: кнопки с популярными городами.

Второй ряд: "Прогноз на завтра".

Третий ряд: "Скорость ветра".

**3. CityWeatherHandler**

Класс для обработки данных о погоде. Отвечает за взаимодействие с API OpenWeatherMap и формирование пользовательских сообщений.

Методы:

getWeatherDescription(city: String, apiKey: String): String  
Получает текущую погоду для заданного города. Возвращает описание с названием города, температурой, погодными условиями и рекомендациями по одежде.

getTomorrowWeather(city: String, apiKey: String): String  
Формирует прогноз погоды на завтра для указанного города, включая температуру и описание (утро, день, вечер, ночь).

getWindInfo(city: String, apiKey: String): String  
Возвращает информацию о скорости и направлении ветра, а также комментарии к погодным условиям.

getSpecificTimeWeather(forecastList: JSONArray, periodName: String, hourTarget: int, isTomorrow: boolean): String  
Извлекает прогноз на определенное время суток (например, утро, день) из данных OpenWeatherMap.

getWindComment(windSpeed: double): String  
Возвращает рекомендации, основанные на скорости ветра (например, "Идеально для прогулки" или "Аккуратно, вас может сдуть").

getWindDirection(degree: int): String  
Конвертирует градусы направления ветра в текстовое описание (например, "Север", "Восток").

**Отношения между классами:**

WeatherBot использует:

CityWeatherHandler для получения данных о погоде.

KeyboardHandler для создания интерфейса.

CityWeatherHandler взаимодействует с OpenWeatherMap API.

KeyboardHandler формирует объекты ReplyKeyboardMarkup, содержащие KeyboardRow и KeyboardButton.

* 1. **Спецификация классов**

#### ****Класс: WeatherBot****

|  |  |
| --- | --- |
| **Поля** | **Описание полей** |
| BOT\_TOKEN | Токен для взаимодействия с Telegram Bot API. |
| BOT\_USERNAME | Имя бота в Telegram. |
| WEATHER\_API\_KEY | API-ключ для доступа к OpenWeatherMap. |
| userStates | Карта для хранения текущего состояния пользователей (ID чата -> выбранный город). |

|  |  |
| --- | --- |
| **Методы** | **Описание Методов** |
| main(String[] args) | Запускает бота, регистрирует его через TelegramBotsApi. |
| onUpdateReceived(Update update) | Обрабатывает входящие обновления и сообщения от пользователей. |
| processMessage(String chatId, String userMessage) | Определяет ответ на сообщение пользователя. Обновляет состояние пользователя и вызывает соответствующий метод из CityWeatherHandler. |
| getBotUsername() | Возвращает имя бота. |
| getBotToken() | Возвращает токен бота. |

#### ****Класс: KeyboardHandler****

|  |  |
| --- | --- |
| **Поля** | **Описание полей** |
| Нет полей | Класс не имеет полей. |

|  |  |
| --- | --- |
| **Методы** | **Описание Методов** |
| getKeyboard() | Возвращает объект ReplyKeyboardMarkup, содержащий предустановленные кнопки для удобного взаимодействия с пользователем. |

**Класс: CityWeatherHandler**

|  |  |
| --- | --- |
| **Поля** | **Описание полей** |
| Нет полей | Класс не имеет полей. |

|  |  |
| --- | --- |
| **Методы** | **Описание Методов** |
| getWeatherDescription(String city, String apiKey) | Получает и возвращает текущую погоду (описание, температура, советы по одежде) для указанного города. |
| getTomorrowWeather(String city, String apiKey) | Формирует прогноз погоды на завтра для указанного города. |
| getWindInfo(String city, String apiKey) | Возвращает информацию о скорости и направлении ветра. |
| getSpecificTimeWeather(JSONArray forecastList, String periodName, int hourTarget, boolean isTomorrow) | Извлекает прогноз погоды на определенное время суток из данных API. |
| getWindComment(double windSpeed) | Возвращает рекомендации, основанные на скорости ветра. |
| getWindDirection(int degree) | Преобразует градусы направления ветра в текстовое описание (например, "Север", "Восток"). |

* 1. **Код классов объектной модели**

**Класс: WeatherBot**

public class WeatherBot extends TelegramLongPollingBot {  
 private static final String *BOT\_TOKEN* = "—";  
 private static final String *BOT\_USERNAME* = "Weather\_W1se\_Bot";  
 private static final String *WEATHER\_API\_KEY* = "c215c7d171aa0faf347de95d36f87980";  
 private final Map<String, String> userStates = new HashMap<>();  
 public static void main(String[] args) {  
 try {  
 TelegramBotsApi botsApi = new TelegramBotsApi(DefaultBotSession.class);  
 botsApi.registerBot(new WeatherBot());  
 System.*out*.println("Бот запущен.");  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
 public void onUpdateReceived(Update update) {  
 if (update.hasMessage()) {  
 if (update.getMessage().hasText()) {  
 String userMessage = update.getMessage().getText();  
 String chatId = update.getMessage().getChatId().toString();  
 String userName = update.getMessage().getFrom().getUserName();  
 System.*out*.println("Пользовательский запрос: [" + userName + "] " + userMessage);  
 String response = processMessage(chatId, userMessage);  
 SendMessage message = new SendMessage();  
 message.setChatId(chatId);  
 message.setText(response);  
 ReplyKeyboardMarkup keyboardMarkup = KeyboardHandler.*getKeyboard*();  
 message.setReplyMarkup(keyboardMarkup);  
 try {  
 execute(message);  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
  
 } else {  
 SendMessage message = new SendMessage();  
 message.setChatId(update.getMessage().getChatId().toString());  
 message.setText("Сообщение не распознано. Пожалуйста, отправьте текст.");  
  
 try {  
 execute(message);  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 private String processMessage(String chatId, String userMessage) {  
 if (userMessage.equals("/start")) {  
 userStates.put(chatId, null);  
 return "Привет! Я бот, который помогает узнать погоду. Напишите название города для прогноза.";  
 }  
 if (userMessage.equals("Прогноз на завтра")) {  
 String currentCity = userStates.get(chatId);  
 if (currentCity == null) {  
 return "Сначала введите название города, чтобы получить прогноз на завтра.";  
 }  
 return CityWeatherHandler.*getTomorrowWeather*(currentCity, *WEATHER\_API\_KEY*);  
 }  
 else if (userMessage.equals("Скорость ветра")){  
 String currentCity = userStates.get(chatId);  
 if (currentCity == null) {  
 return "Сначала введите название города, чтобы получить скорость ветра.";  
 }  
 return CityWeatherHandler.*getWindInfo*(currentCity, *WEATHER\_API\_KEY*);  
 }  
 userStates.put(chatId, userMessage);  
 return CityWeatherHandler.*getWeatherDescription*(userMessage, *WEATHER\_API\_KEY*);  
 }  
 public String getBotUsername() {  
 return *BOT\_USERNAME*;  
 }  
 public String getBotToken() {  
 return *BOT\_TOKEN*;  
 }  
}

#### ****Класс: KeyboardHandler****

class KeyboardHandler {  
 public static ReplyKeyboardMarkup getKeyboard() {  
 ReplyKeyboardMarkup keyboardMarkup = new ReplyKeyboardMarkup();  
 List<KeyboardRow> keyboard = new ArrayList<>();  
 KeyboardRow row1 = new KeyboardRow();  
 row1.add(new KeyboardButton("Москва"));  
 row1.add(new KeyboardButton("Санкт-Петербург"));  
 row1.add(new KeyboardButton("Екатеринбург"));  
 KeyboardRow row2 = new KeyboardRow();  
 row2.add(new KeyboardButton("Прогноз на завтра"));  
 KeyboardRow row3 = new KeyboardRow();  
 row3.add(new KeyboardButton("Скорость ветра"));  
 keyboard.add(row1);  
 keyboard.add(row2);  
 keyboard.add(row3);  
 keyboardMarkup.setKeyboard(keyboard);  
 keyboardMarkup.setResizeKeyboard(true);  
 return keyboardMarkup;  
 }  
}

#### ****Класс: CityWeatherHandler****

class CityWeatherHandler {  
 public static String getWeatherDescription(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20"); // Обработка пробелов  
 String apiUrl = "https://api.openweathermap.org/data/2.5/weather?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 String cityName = jsonResponse.getString("name");  
 JSONObject sysInfo = jsonResponse.getJSONObject("sys");  
 String country = sysInfo.getString("country");  
 String description = jsonResponse.getJSONArray("weather").getJSONObject(0).getString("description");  
 double temperature = jsonResponse.getJSONObject("main").getDouble("temp");  
 String clothingAdvice = *getClothingAdvice*(temperature, description);  
 return "Город: " + cityName + " (" + country + ")\n" +  
 "Описание: " + description + "\n" +  
 "Температура: " + temperature + "°C\n" +  
 clothingAdvice;  
 } catch (Exception e) {  
 return "Не удалось получить данные для города: " + city + ". Проверьте правильность написания.";  
 }  
 }  
 private static String getClothingAdvice ( double temperature, String description){  
 StringBuilder advice = new StringBuilder("Советы по одежде:\n");  
 if(temperature <= -20){  
 advice.append("- Наденьте тёплую куртку, шапку и шарф.\n");  
 advice.append("- Возьмите перчатки или варежки.\n");  
 advice.append("- Лучше оставайтесь дома.\n");  
 }  
 else if (temperature <= -10) {  
 advice.append("- Наденьте тёплую куртку, шапку и шарф.\n");  
 advice.append("- Возьмите перчатки или варежки.\n");  
 }else if (temperature<0) {  
 advice.append("- Наденьте куртку потеплее и не забудьте шапку.\n");  
 }else if (temperature <= 10) {  
 advice.append("- Подойдёт осенняя куртка или пальто.\n");  
 advice.append("- Возможно, вам понадобится лёгкая шапка.\n");  
 } else if (temperature <= 20) {  
 advice.append("- Лёгкая куртка или толстовка будут кстати.\n");  
 } else {  
 advice.append("- Наденьте футболку и лёгкие брюки или шорты.\n");  
 advice.append("- Не забудьте головной убор, если солнечно.\n");  
 }  
 if (description.contains("дождь") || description.contains("ливень")) {  
 advice.append("- Возьмите зонт или дождевик.\n");  
 advice.append("- Наденьте непромокаемую обувь.\n");  
 } else if (description.contains("снег")) {  
 advice.append("- Наденьте обувь с нескользящей подошвой.\n");  
 }  
 return advice.toString();  
 }  
  
 private static String getSpecificTimeWeather(JSONArray forecastList, String periodName, int hourTarget, boolean isTomorrow) {  
 for (int i = 0; i < forecastList.length(); i++) {  
 JSONObject forecast = forecastList.getJSONObject(i);  
 String dateText = forecast.getString("dt\_txt");  
 String[] dateParts = dateText.split(" ");  
 int hour = Integer.*parseInt*(dateParts[1].split(":")[0]);  
 boolean isToday = dateParts[0].equals(java.time.LocalDate.*now*().toString());  
 boolean isForTomorrow = dateParts[0].equals(java.time.LocalDate.*now*().plusDays(1).toString());  
 if ((isTomorrow && isForTomorrow || !isTomorrow && isToday) && hour == hourTarget) {  
 String description = forecast.getJSONArray("weather").getJSONObject(0).getString("description");  
 double temperature = forecast.getJSONObject("main").getDouble("temp");  
 return periodName + ": " + description + ", " + temperature + "°C\n";  
 }  
 }  
 return periodName + ": данные отсутствуют.\n";  
 }  
  
 public static String getTomorrowWeather(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20");  
 String apiUrl = "https://api.openweathermap.org/data/2.5/forecast?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 JSONArray forecastList = jsonResponse.getJSONArray("list");  
 StringBuilder result = new StringBuilder("Прогноз на завтра для города " + city + ":\n");  
 result.append(*getSpecificTimeWeather*(forecastList, "Ночь", 3, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "Утро", 9, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "День", 15, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "Вечер", 21, true));  
 return result.toString();  
 } catch (Exception e) {  
 return "Не удалось получить прогноз на завтра для города: " + city + ". Проверьте правильность написания.";  
 }  
 }  
 public static String getWindInfo(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20");  
 String apiUrl = "https://api.openweathermap.org/data/2.5/weather?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 double windSpeed = jsonResponse.getJSONObject("wind").getDouble("speed");  
 int windDegree = jsonResponse.getJSONObject("wind").getInt("deg");  
 String windDirection = *getWindDirection*(windDegree);  
 String windComment = *getWindComment*(windSpeed);  
 return "Скорость ветра: " + windSpeed + " м/с\n" +  
 "Направление ветра: " + windDirection+ "\n" +  
 windComment;  
 } catch (Exception e) {  
 return "Не удалось получить данные о ветре для города " + city + ".";  
 }  
 }  
 private static String getWindComment(double windSpeed) {  
 if (windSpeed <= 1.7) {  
 return "Идеально для прогулки — ветра почти нет!";  
 } else if (windSpeed <= 3.3) {  
 return "Легкий ветерок — наслаждайтесь свежим воздухом.";  
 } else if (windSpeed <= 7.4) {  
 return "Держите шляпу, чтобы не улетела!";  
 } else if (windSpeed <= 12.4) {  
 return "Ветерок уже чувствуется — крепче держитесь за поручни!";  
 } else {  
 return "Аккуратно, вас может сдуть!";  
 }  
 }  
 private static String getWindDirection(int degree) {  
 String[] directions = {"Север", "Северо-восток", "Восток", "Юго-восток",  
 "Юг", "Юго-запад", "Запад", "Северо-запад"};  
 return directions[(int)Math.*round*((degree % 360) / 45.0) % 8];  
 }  
}

* 1. **Интерфейс программы**

Пользовательский интерфейс программы представляет собой бота в Telegram (@Weather\_W1se\_Bot).

В начале работы бот встречает пользователя приветственным сообщением:

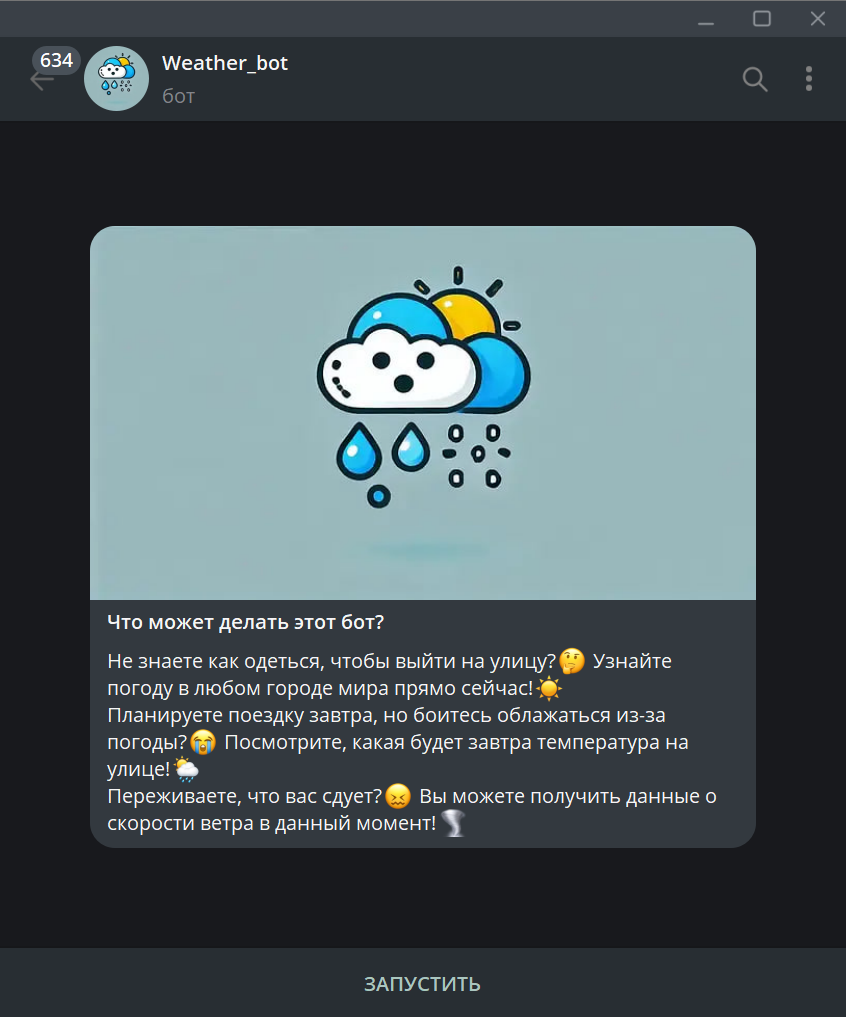


Рис. 3 Начало работы с ботом в Telegram

Нажимаем кнопку «Запустить», при последующей работе за неё отвечает команда «/start». Нас встречает сообщение с предложением узнать прогноз погоды, введя запрос самостоятельно или выбрав команду по кнопке. Рассмотрим разные варианты взаимодействия с ботом.

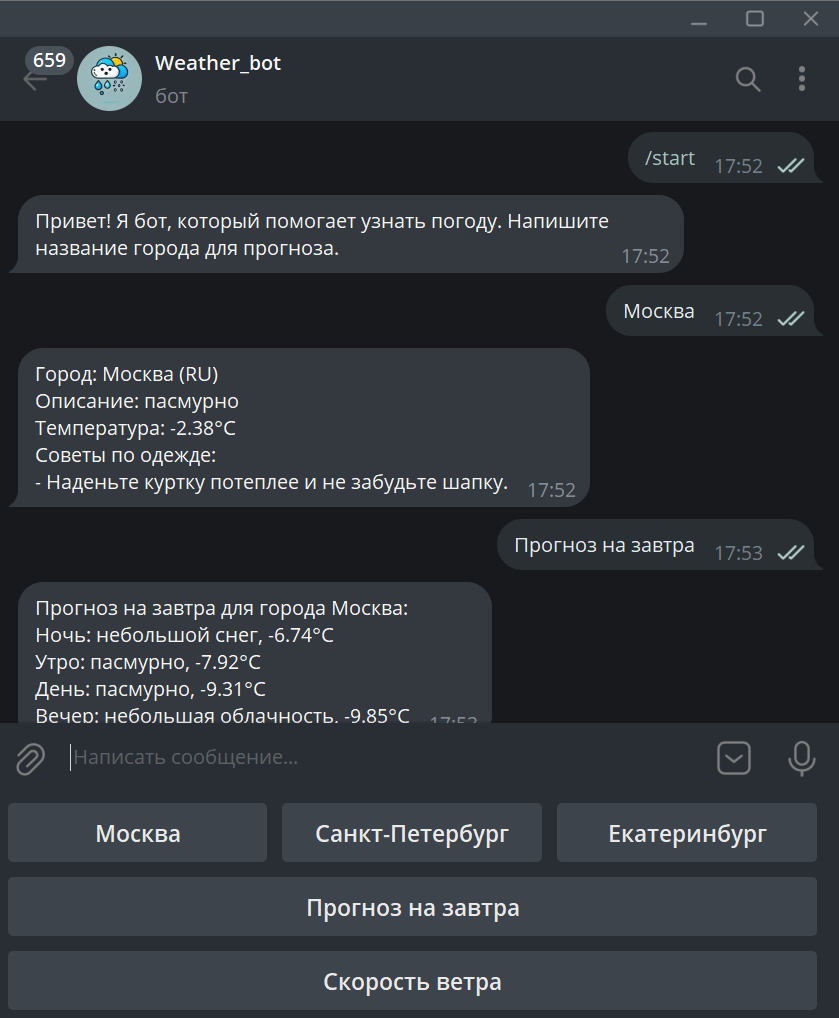


Рис. 4 Использование готовых команд в боте

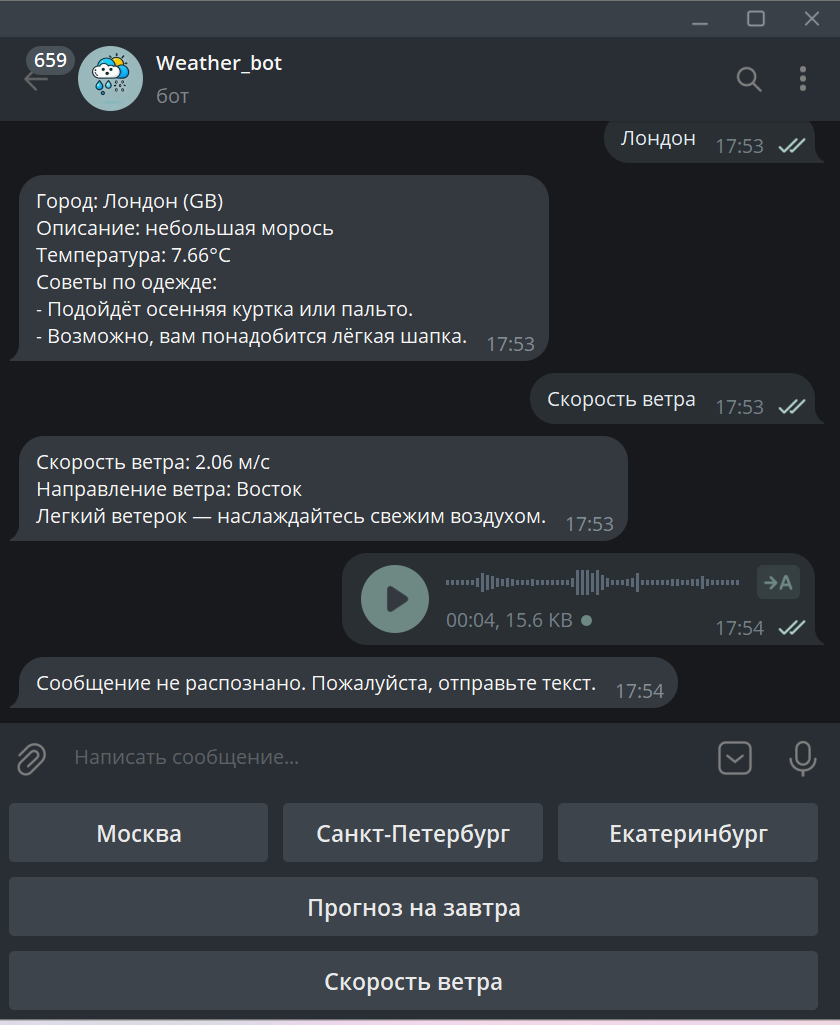


Рис. 5 Ввод своего запроса боту и обработка не текстовых сообщений

Как видим на рисунке 5, бот может отвечать не только по готовым командам, но и принимать названия городов, которые пользователь введёт самостоятельно, после чего по ним обрабатывать остальные команды. При вводе несуществующего города (или любого другого слова), бот корректно отреагирует, попросив ввести новый запрос. А если по каким-либо причинам сайт не сможет вернуть информацию о выбранном городе, то бот укажет на ошибку обработки запроса.

**ЗАКЛЮЧЕНИЕ**

В рамках выполнения данной работы был разработан телеграм-бот для предоставления информации о погоде с использованием API OpenWeatherMap. Бот успешно выполняет основные функции:

**Разработка удобного интерфейса взаимодействия**:

Был реализован пользовательский интерфейс с помощью кнопок клавиатуры (ReplyKeyboardMarkup), что упрощает взаимодействие с ботом.

**Реализация получения данных о погоде**:

Бот интегрирован с OpenWeatherMap API для получения информации о текущей погоде, прогнозе на завтра, скорости и направлении ветра. Это позволяет пользователю получать актуальные и подробные сведения.

**Обработка пользовательских запросов**:

Бот обрабатывает команды и текстовые сообщения от пользователей, включая /start, запросы конкретного города или информации о ветре и прогнозе. Состояние пользователя сохраняется для улучшения взаимодействия.

**Интеграция рекомендаций для пользователя**:

Реализована система текстовых советов по выбору одежды и поведению в зависимости от погодных условий (температуры, осадков, ветра). Это повышает полезность бота для повседневного использования.

**Стабильность работы**:

Проект реализован с учетом обработки возможных ошибок, таких как неправильное название города или недоступность данных. Это минимизирует сбои в работе и улучшает пользовательский опыт.

Таким образом, все поставленные цели были достигнуты, а задачи успешно реализованы. Разработанный бот является полезным и удобным инструментом для пользователей, которым необходимо быстро получить информацию о погоде в различных городах.

**Ссылка на репозиторий GitHub**: <https://github.com/Filimonova-Ekaterina/OOP_Coursework/tree/master>

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Паттерны проектирования / Э. Фримен, Э. Робсон. СПб.: Издательство Питер, 2018. 656 с.
2. Погружение в паттерны проектирования / А. Швец. СПб.: Издательство Refactoring.Guru, 2021. 400 с.
3. Всё о чём должен знать разработчик Telegram-ботов //URL: <https://habr.com/ru/articles/543676/> (Дата обращения: 05.12.24)
4. Telegram bot API - разбор документации с примерами // URL: <https://infostart.ru/1c/articles/1217332/> (Дата обращения: 05.12.24)
5. Пример использования Java-библиотеки для создания Telegram-бота // URL: <https://habr.com/ru/articles/432548/> (Дата обращения: 05.12.24)
6. Документация API OpenWeatherMap // URL: <https://openweathermap.org/api/one-call-api> (Дата обращения: 06.12.24)

**ПРИЛОЖЕНИЕ А**

**Листинг**

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.KeyboardButton;  
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;  
import org.telegram.telegrambots.meta.TelegramBotsApi;  
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;  
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.\*;  
import org.json.JSONArray;  
import org.json.JSONObject;  
  
public class WeatherBot extends TelegramLongPollingBot {  
 private static final String *BOT\_TOKEN* = "—";  
 private static final String *BOT\_USERNAME* = "Weather\_W1se\_Bot";  
 private static final String *WEATHER\_API\_KEY* = "c215c7d171aa0faf347de95d36f87980";  
 private final Map<String, String> userStates = new HashMap<>();  
 public static void main(String[] args) {  
 try {  
 TelegramBotsApi botsApi = new TelegramBotsApi(DefaultBotSession.class);  
 botsApi.registerBot(new WeatherBot());  
 System.*out*.println("Бот запущен.");  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
 public void onUpdateReceived(Update update) {  
 if (update.hasMessage()) {  
 if (update.getMessage().hasText()) {  
 String userMessage = update.getMessage().getText();  
 String chatId = update.getMessage().getChatId().toString();  
 String userName = update.getMessage().getFrom().getUserName();  
 System.*out*.println("Пользовательский запрос: [" + userName + "] " + userMessage);  
 String response = processMessage(chatId, userMessage);  
 SendMessage message = new SendMessage();  
 message.setChatId(chatId);  
 message.setText(response);  
 ReplyKeyboardMarkup keyboardMarkup = KeyboardHandler.*getKeyboard*();  
 message.setReplyMarkup(keyboardMarkup);  
 try {  
 execute(message);  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
  
 } else {  
 SendMessage message = new SendMessage();  
 message.setChatId(update.getMessage().getChatId().toString());  
 message.setText("Сообщение не распознано. Пожалуйста, отправьте текст.");  
  
 try {  
 execute(message);  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 private String processMessage(String chatId, String userMessage) {  
 if (userMessage.equals("/start")) {  
 userStates.put(chatId, null);  
 return "Привет! Я бот, который помогает узнать погоду. Напишите название города для прогноза.";  
 }  
 if (userMessage.equals("Прогноз на завтра")) {  
 String currentCity = userStates.get(chatId);  
 if (currentCity == null) {  
 return "Сначала введите название города, чтобы получить прогноз на завтра.";  
 }  
 return CityWeatherHandler.*getTomorrowWeather*(currentCity, *WEATHER\_API\_KEY*);  
 }  
 else if (userMessage.equals("Скорость ветра")){  
 String currentCity = userStates.get(chatId);  
 if (currentCity == null) {  
 return "Сначала введите название города, чтобы получить скорость ветра.";  
 }  
 return CityWeatherHandler.*getWindInfo*(currentCity, *WEATHER\_API\_KEY*);  
 }  
 userStates.put(chatId, userMessage);  
 return CityWeatherHandler.*getWeatherDescription*(userMessage, *WEATHER\_API\_KEY*);  
 }  
 public String getBotUsername() {  
 return *BOT\_USERNAME*;  
 }  
 public String getBotToken() {  
 return *BOT\_TOKEN*;  
 }  
}  
  
class KeyboardHandler {  
 public static ReplyKeyboardMarkup getKeyboard() {  
 ReplyKeyboardMarkup keyboardMarkup = new ReplyKeyboardMarkup();  
 List<KeyboardRow> keyboard = new ArrayList<>();  
 KeyboardRow row1 = new KeyboardRow();  
 row1.add(new KeyboardButton("Москва"));  
 row1.add(new KeyboardButton("Санкт-Петербург"));  
 row1.add(new KeyboardButton("Екатеринбург"));  
 KeyboardRow row2 = new KeyboardRow();  
 row2.add(new KeyboardButton("Прогноз на завтра"));  
 KeyboardRow row3 = new KeyboardRow();  
 row3.add(new KeyboardButton("Скорость ветра"));  
 keyboard.add(row1);  
 keyboard.add(row2);  
 keyboard.add(row3);  
 keyboardMarkup.setKeyboard(keyboard);  
 keyboardMarkup.setResizeKeyboard(true);  
 return keyboardMarkup;  
 }  
}  
  
class CityWeatherHandler {  
 public static String getWeatherDescription(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20"); // Обработка пробелов  
 String apiUrl = "https://api.openweathermap.org/data/2.5/weather?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 String cityName = jsonResponse.getString("name");  
 JSONObject sysInfo = jsonResponse.getJSONObject("sys");  
 String country = sysInfo.getString("country");  
 String description = jsonResponse.getJSONArray("weather").getJSONObject(0).getString("description");  
 double temperature = jsonResponse.getJSONObject("main").getDouble("temp");  
 String clothingAdvice = *getClothingAdvice*(temperature, description);  
 return "Город: " + cityName + " (" + country + ")\n" +  
 "Описание: " + description + "\n" +  
 "Температура: " + temperature + "°C\n" +  
 clothingAdvice;  
 } catch (Exception e) {  
 return "Не удалось получить данные для города: " + city + ". Проверьте правильность написания.";  
 }  
 }  
 private static String getClothingAdvice ( double temperature, String description){  
 StringBuilder advice = new StringBuilder("Советы по одежде:\n");  
 if(temperature <= -20){  
 advice.append("- Наденьте тёплую куртку, шапку и шарф.\n");  
 advice.append("- Возьмите перчатки или варежки.\n");  
 advice.append("- Лучше оставайтесь дома.\n");  
 }  
 else if (temperature <= -10) {  
 advice.append("- Наденьте тёплую куртку, шапку и шарф.\n");  
 advice.append("- Возьмите перчатки или варежки.\n");  
 }else if (temperature<0) {  
 advice.append("- Наденьте куртку потеплее и не забудьте шапку.\n");  
 }else if (temperature <= 10) {  
 advice.append("- Подойдёт осенняя куртка или пальто.\n");  
 advice.append("- Возможно, вам понадобится лёгкая шапка.\n");  
 } else if (temperature <= 20) {  
 advice.append("- Лёгкая куртка или толстовка будут кстати.\n");  
 } else {  
 advice.append("- Наденьте футболку и лёгкие брюки или шорты.\n");  
 advice.append("- Не забудьте головной убор, если солнечно.\n");  
 }  
 if (description.contains("дождь") || description.contains("ливень")) {  
 advice.append("- Возьмите зонт или дождевик.\n");  
 advice.append("- Наденьте непромокаемую обувь.\n");  
 } else if (description.contains("снег")) {  
 advice.append("- Наденьте обувь с нескользящей подошвой.\n");  
 }  
 return advice.toString();  
 }  
  
 private static String getSpecificTimeWeather(JSONArray forecastList, String periodName, int hourTarget, boolean isTomorrow) {  
 for (int i = 0; i < forecastList.length(); i++) {  
 JSONObject forecast = forecastList.getJSONObject(i);  
 String dateText = forecast.getString("dt\_txt");  
 String[] dateParts = dateText.split(" ");  
 int hour = Integer.*parseInt*(dateParts[1].split(":")[0]);  
 boolean isToday = dateParts[0].equals(java.time.LocalDate.*now*().toString());  
 boolean isForTomorrow = dateParts[0].equals(java.time.LocalDate.*now*().plusDays(1).toString());  
 if ((isTomorrow && isForTomorrow || !isTomorrow && isToday) && hour == hourTarget) {  
 String description = forecast.getJSONArray("weather").getJSONObject(0).getString("description");  
 double temperature = forecast.getJSONObject("main").getDouble("temp");  
 return periodName + ": " + description + ", " + temperature + "°C\n";  
 }  
 }  
 return periodName + ": данные отсутствуют.\n";  
 }  
  
 public static String getTomorrowWeather(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20");  
 String apiUrl = "https://api.openweathermap.org/data/2.5/forecast?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 JSONArray forecastList = jsonResponse.getJSONArray("list");  
 StringBuilder result = new StringBuilder("Прогноз на завтра для города " + city + ":\n");  
 result.append(*getSpecificTimeWeather*(forecastList, "Ночь", 3, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "Утро", 9, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "День", 15, true));  
 result.append(*getSpecificTimeWeather*(forecastList, "Вечер", 21, true));  
 return result.toString();  
 } catch (Exception e) {  
 return "Не удалось получить прогноз на завтра для города: " + city + ". Проверьте правильность написания.";  
 }  
 }  
 public static String getWindInfo(String city, String apiKey) {  
 String formattedCity = city.trim().replace(" ", "%20");  
 String apiUrl = "https://api.openweathermap.org/data/2.5/weather?q=" + formattedCity +  
 "&appid=" + apiKey + "&units=metric&lang=ru";  
 try {  
 URL url = new URL(apiUrl);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
 while ((line = in.readLine()) != null) {  
 response.append(line);  
 }  
 in.close();  
 JSONObject jsonResponse = new JSONObject(response.toString());  
 double windSpeed = jsonResponse.getJSONObject("wind").getDouble("speed");  
 int windDegree = jsonResponse.getJSONObject("wind").getInt("deg");  
 String windDirection = *getWindDirection*(windDegree);  
 String windComment = *getWindComment*(windSpeed);  
 return "Скорость ветра: " + windSpeed + " м/с\n" +  
 "Направление ветра: " + windDirection+ "\n" +  
 windComment;  
 } catch (Exception e) {  
 return "Не удалось получить данные о ветре для города " + city + ".";  
 }  
 }  
 private static String getWindComment(double windSpeed) {  
 if (windSpeed <= 1.7) {  
 return "Идеально для прогулки — ветра почти нет!";  
 } else if (windSpeed <= 3.3) {  
 return "Легкий ветерок — наслаждайтесь свежим воздухом.";  
 } else if (windSpeed <= 7.4) {  
 return "Держите шляпу, чтобы не улетела!";  
 } else if (windSpeed <= 12.4) {  
 return "Ветерок уже чувствуется — крепче держитесь за поручни!";  
 } else {  
 return "Аккуратно, вас может сдуть!";  
 }  
 }  
 private static String getWindDirection(int degree) {  
 String[] directions = {"Север", "Северо-восток", "Восток", "Юго-восток",  
 "Юг", "Юго-запад", "Запад", "Северо-запад"};  
 return directions[(int)Math.*round*((degree % 360) / 45.0) % 8];  
 }  
}