EcoWeather App

AfzoodNaaz Surati | GH1028962

B202 Advance Programming

1/14/25

GitHub Repository Link: https://github.com/afzoodnaaz/EcoWeather.git

[Grab your reader's attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

Table of Contents

Introduction:

The EcoWeather App is a Java-based desktop application designed to provide real time weather figures for both, user specified cities and crocodile habitats around the world. This application is an extension to my previous project ReptiLink, a desktop-based application that operates as a conservational tool for the management of reptiles, specifically crocodilians using geolocation, logging, and pattern prediction algorithms. The motivation behind creating this project was to demonstrate proficiency in Object Oriented Programming (OOP) and programming a fully functional, comprehensible tool that can be used for assessing environmental information in addition to conservational awareness and habitat monitoring.

Crocodiles reside in diverse eco systems across multiple continents and weather conditions in these habitat s heavily influence their behaviours, breeding patterns, and conservational strategies. This project aims to be a functional addition to conservational sciences, hence delivering a holistic educational application. EcoWeather is brought to life by the combination of weather APIs with geographic data (latitude and longitude) and a graphical user interface (GUI).

The primary goals for the EcoWeather App project include:

- Implementing Object Oriented Programming principles, including classes, interfaces, encapsulation, and even modular design.
- Successfully demonstrating the use of API integration in Java for accurate data retrieval.
- Developing a user-friendly GUI that separates city-based weather, acting as a simple weather app, and habitat-based weather for a specific user demographic.
- Providing accurate weather information for all crocodile habitats across the world using intensive research and proper linking of exact geolocations with weather APIs.
- Creating a solid base within this application to lay the foundation for future improvements including weather data and it's influences for animal and plant species, all comprised in one application.

The EcoWeather app is built using Java Swing GUI for easy user interaction, with in turn fetches current weather data by communicating to two APIs: Weather API and Location API. This project aims to emphasize a clean and modular code in order to make it maintainable and scalable.

This report outlines the project's developmental stages, highlighting the system architecture and its implementation, with a comprehensive overview of the results and the challenges faced during its development. As this project is a simple prototype for an addition to an existing conservational tool, there are multiple recommendations in regards to the expansion and improvement of this project in order to create a whole system of conservational tools highlighting every aspect of this field.

System Architecture:

The system architecture of the EcoWeather app adheres to Object Oriented Programming principles and follows a modular approach. There are several core components to this application and are divided based on functionality.

1. Main class (Main.java)

   The Main.java serves as an entry point for this project by initialising objects of services like WeatherService (ws), GeoLocationService (gs), and HabitatWeatherService (hs). The Main class then launches the GUI by passing the service objects that were first initialised. The modular approach allows this project to have a clean, decluttered Main class just used for calling other java classes in a coherent way.

2. User interface (GUI.java)

   The graphical user interface is designed using Java Swing and provides a comprehensive experience using two main pages (sections), City Weather for normal weather information and Habitat Weather for specified information on crocodilian habitat weather conditions. Both these section display almost the same information; Location, Temperature in degree Celsius, Humidity, and Wind speed. Although the Habitat Weather provides additional information about the influence of the area's weather conditions on the specified crocodilian species inhabited there. The GUI.java class uses JTabbedPane to separate the two sections in two tabs for intuitive navigation. The City Weather section comprises of a TextBox for the user's input, a "Get Weather" push button to call the function to retrieve information in correspondence with the input, and displays the output in a scrollable JTextArea. The Habitat Weather comprises of a dropdown list of Habitats currently in the system for the user to choose from, a "Get Weather" push button, and similarly displays the output in a scrollable JTextArea. The entire GUI also incorporates GridBadLayout for a clean and responsive placement of each component.

3. Geolocation service (GeoLocationService.java)

   This class retrieves specific geographic location coordinates for the user inputted cities using the geocoding API https://open-meteo.com/en/docs/geocoding-api. The use of this API ensures accurate location based data for the functionality of this project. The data retrieved in this class directly corresponds to WeatherService.java for correct real-time weather data.

4. Weather service (WeatherService.java)

   This class handles the API calls to Open-Meteo's Weather forecast API, using the latitude and longitude provided by GeoLocationServices.java. The exact temperature, humidity, and windspeed are extracted in this class as a response to parsing JSON. Finally, providing us with a centralised location for all of its weather-related computations.

5. Habitat weather service (HabitatWeatherService.java)

   HabitatWeatherService.java essentially performs the way as WeatherService.java but for predefined crocodilian habitats from the data class HabitatData.java. It uses WeatherService.java to retrieve weather information for each habitat as well as additional information linked to species behaviour residing in said habitat. Its final functionality is returning an object of class HabitatWeatherData containing the habitat name and its weather details. This is the class that makes the EcoWeather App stand out as opposed to other regular weather service applications. Given the current limitations of the habitat data in this moment, it makes for a unique addition to crocodilian conservational tools.

6. Data classes (WeatherData.java | HabitatWeatherData.java | HabitatData.java)
    - WeatherData.java stores weather information of the user inputted city retrieved using WeatherService.java and it's API connection and maintains separation of data and its presentation.
    - HabitatWeatherData.java stores weather information of the user selected habitat retrieved using HabitatWeatherService.java and it's API connection. Similarly to WeatherData.java it also maintains a clean serpartion of weather data and its presentation.
    - HabitatData.java stores the habitat information, along with their coordinates, crocodile species, and behavioural data, providing a list of the GUI dropdown selection. This information is accurately researched and manually stored by the developer (me) with hopes to expand and be functional for user inputted habitat locations.
7. Utility Classes (HttpUtil.java | JsonUtil.java)
    - HttpUtil.java handles the HttpURLConnections and the HTTP GET requests and retrieves the raw JSON responses while handling exceptions.
    - JsonUtil.java parses the JSON strings to JSONObjects for smooth extractions of data fields to prove an overall effect system.

This architecture ensures a separation of each component's responsibilities, which intern improves maintainability, readability, and future scalability. This provides a strong base for adding new habitats for additional animal species with intensive weather parameters. This would require modifications in a few specific classes rather than across the entire codebase.

Implementation

The implementation process was fairly simple as it was broken down in five main stages starting with brainstorming, designing the project, coding using Java, testing and finally, integration. Each stage applies core Java-specific features to the application.

Object Oriented Design

EcoWeather App uses Object Oriented Programming principles extensively throughout the project, such as:

- Encapsulation: The WeatherData class and HabitatWeatherData class use encapsulation with the help of getter and setter methods.
- Inheritance: Inheritance is used in this application throughout the Habitat Data types defined during development. Habitat.java is the super class and several Habitat subclasses like WetlandsHabitat.java, MangroovesHabitat. Java use extends key words to access shared parent class types.
- Polymorphism: This is also used throughout the program to create different types of habitat classes that share a common structure while returning different weather information; description and coordinates.
- Modularity: All service classes like GeoLocationService, WeatherService and HabitatWeatherSrvice are completely independent of each other as modules and handle specific functionalities within themselves.
- Reusability: The utility classes are designed with intention of being reused across multiple applications and services.

- Abstraction: This makes EcoWeather as an application user friendly, since users only interact with the GUI without having an understanding of the API calls and JSON parsing.

EcoWeather showcases all the OOP practices in an organised manner and is a great example object-oriented design principles can be used by a developer to build a clean and modular Java application with external API integration.

API Integration

In order to create a fully functional Weather app, development highly relies of the Open-Meteo APIs for geolocation data and weather data. These free APIs provide access to real-time weather, temperature, humidity, and wind speed in correspondence to the location provided by the user. These APIs were integrated in WeatherService.java and GeoLocationService.java.

1. The API URLs were generated based on user preferences and on user input, i.e. city name, habitat name (list provided by the developer for prototype).
2. The data from the API is then fetched using the utility class HttpUtil.java with the help of the HTTP GET request.
3. This application parses the JSON response using JSONParser from the orj.json.simple imported library.
4. The data fields such as temperature (within 2 m), humidity (within 2 m), and wind speed (within 10 m) are then extracted to be store in the Data classes.

The separation of the use of JSON parsing and HTTP utility ensures the reusability and efficiency of the code.

GUI Design

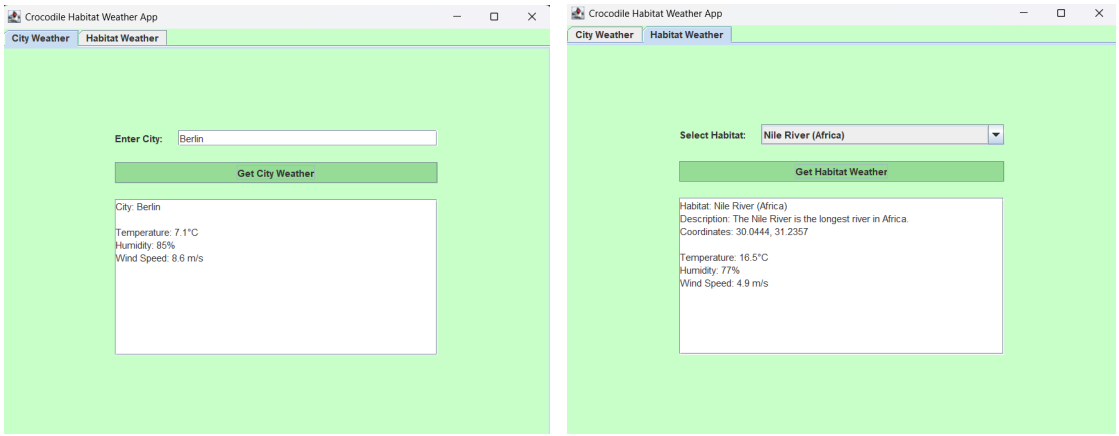
The GUI design was heavily influences by user surveys with elements inspired by the educational and conservational theme. This design emphasises clarity and positive user experience, as well as usability.

Non-functional design choices include the background, colour, and page layout. A light green background was chosen to reflect the environmental and reptile theme while portraying a professional look.

The GUI is split into two tabs: City Weather and Habitat Weather.

- City Weather Tab: This page includes a text area for the user to enter the city name, followed by a button underneath to fetch the weather data corresponding to the user's input. The output is then displayed below on a scorable TextArea with the City name, temperature, humidity, and wind speed.
- Habitat Weather Tab: This page includes a drop-down list of habitats across the world for users to select from, followed by a button underneath to fetch the weather data corresponding to the user's input. The output is then displayed below on a scorable TextArea with the Habitat name and location, habitat description, coordinates, temperature, humidity, and wind speed.

The user interface directly interacts with the defined services to retrieve and display data which in turn demonstrates the MCV concept (Model view controller).



Exception Handling

Handling exceptions is a crucial part of this application since it's functionality heavily dependant on the user. Therefore, multiple potential failure points were addressed during development.

- Invalid user inputs: Parameters were put in place, so all exceptions are caught and reported to the user through the GUI, giving the user to rectify and enter the correct input.
- API connection: The HttpUtil class checks the HTTP response codes to ensure successful and API connections and detect and handle unsuccessful ones.
- JSON parsing: Exceptions are caught in JsonUtil.java to prevent the application from completely crashing.

These exception handling placements interact with the users, giving them feedback, while also maintaining stability throughout the application.

Habitat Data

In order to create the first prototype of this application, crocodile habitat data needed to be fetched and researched during the developmental process so as to include a predefined set of geographically correct habitats.

No.	Habitat	Coordinates
1	Nile River (Africa)	(30.044, 31.235)
2	Queensland (Australia)	(-19.257, 146.818)
3	Florida Everglades (USA)	(25.286, -80.898)
4	Orinoco Basin (South America)	(4.0, -67.0)
5	Southeast Asia Mangroves	(1.352, 103.819)
6	Okavango Delta (Botswana)	(-19.133, 23.583)
7	Ganges River (India)	(25.594, 85.137)
8	Mekong Delta (Vietnam)	(10.033, 105.783)
9	Cameroon Wetlands (Cameroon)	(4.0, 12.0)
10	Lake Argyle (Australia)	(15.383, 128.733)
11	Everglades National Park (USA)	(25.286, -80.898)
12	Amazon Basin (South America)	(-3.465, -62.215)
13	Indus River (Pakistan)	(30.157, 71.524)
14	Yangtze River (China)	(31.230, 121.473)
15	Madagascar Wetlands (Madagascar)	(18.7669, 46.869)

These approximate coordinates are fed to the HabitatWeatherService class to fetch data via the Open-Meteo Weather API for each habitat, allowing the users to explore educational aspects of areas inhabited by crocodilians.

Results

The EcoWeather app successfully retrieves and displays weather information for user specified cities and crocodile habitats. The GUI accurately and constantly updates the program based on the user's input with no input number limitations.

Challenges and Solutions

The EcoWeather App is not an extremely complex program, but a few challenges needed to be tackled to successfully host the application's functionality.

1. API limitations:

Challenge: The main challenge I encountered during the developmental stage of the EcoWeather application was the limitations of the location and weather API, especially when it comes making quick and repeated requests in a short time frame. In the testing stage, uninterrupted weather data searches, especially while constantly switching between the City weather search and Habitat weather search, frequently resulted in delayed response times and inaccurate results.

Solution: This application was designed in a way to minimize any unnecessary API calls exclusively triggering requests based on user actions, specificcaly clicking the button to fetch weather data.

2. JSON parsing:

Challenge: Parsing the JSON response received by the weather API was another significant challenge in the development of EcoWeather. The API provided a structured data which contained nested objects, which was qyite difficult to locate and extract for desired outputs. This issue also led to runtime errors and incomplete data retrieval.

Solution: The JsonUtility class was introduced to solely deal with the parsing logic that allowed all JSON operations to be handled in a centralised location. By making this separation, any future updates to the API response structure can be applied without any extensive refactoring.

3. GUI responsiveness:

Challenge: Comprising all the components of the EcoWeather App into a comprehensive user interfaced seemed to pose a whole new challenge, especially while using Java Swing. The GridBagLayout offered a high degree of flexibility but also required a good amount of configuration via constraints to achieve optimal spacing and sizing.

Solution: To merge the two-tab design, ensuring the interface's consistency with alignment, constraint values were adjusted throughout the GUI to improve balance and readability. BorderLayout was integrated during the early developmental stage, but GridBagLayout was reenforced given its adaptability and control over placement.

Future Work

The EcoWeather Application as opposed to being a stand-alone weather application, can also be expanded into being a component of a broader digital conservational tool. The current system just focuses on environmental weather data for specified locations and habitats, and its functionality can add to the foundation of a complex yet comprehensive system of tools surrounding wildlife and plant conservation. This platform of tools can be expanded into integrating data on species behaviour, habitat conditions, breeding and mating seasons, migration patterns, and human impact, along with weather influences on species and habits. This analysis of factors such as weather patterns and other components could bring this application to life with intent to support conservational awareness, educational research, and future decisions to maintain a healthy ecosystem. Therefore, EcoWeather acts as a building block to an interconnected system designed to educate and contribute to long term environmental sustainability.

For the near future, potential improvements include:

- Adding an interactive map, inspired by current day weather apps and https://databayou.com/crocodile/habitat.html, where users can visualise and interact with habitats on a live map with weather overlays.
- Expanding weather data fetched by the app including future forecasts, UV index, air quality, and weather information with respect to local area times.
- Broadening the applications scope as opposed to just being a desktop-based application, by potentially porting the application to Android and Apple platforms (mobile adaptation).
- Filtering regularly used cities/habitats based on user preference. i.e. adding a set of 'favourites' in a section where users have live outputs without having to manually input required cities/habitats.

Conclusion

The EcoWeather App is developed to represent all the Object Oriented Programming principles in Java while highlighting the integration of real-world functions like API calling, Jsong paring, and GUI development. This app provides weather data on various cities and crocodile habitats with future plans to expand into all wildlife and plant life habitats. This project offers a fun, yet interactive and informative platform for the optimal user experience by marrying practical functionalities, and educational and conservative content. Overall, the EcoWeather Application successfully meets all its primary goals by bridging the gap between ecological knowledge and practical weather data to create a functional and informative conservational tool.

References

GeoLocation API:

https://open-meteo.com/en/docs/geocoding-api

Weather API:

https://open-meteo.com/en/docs

API Connections:

https://github.com/curadProgrammer

Crocodile Habitat data:

https://education.nationalgeographic.org/resource/crocodilian-ranges/

https://databayou.com/crocodile/habitat.html