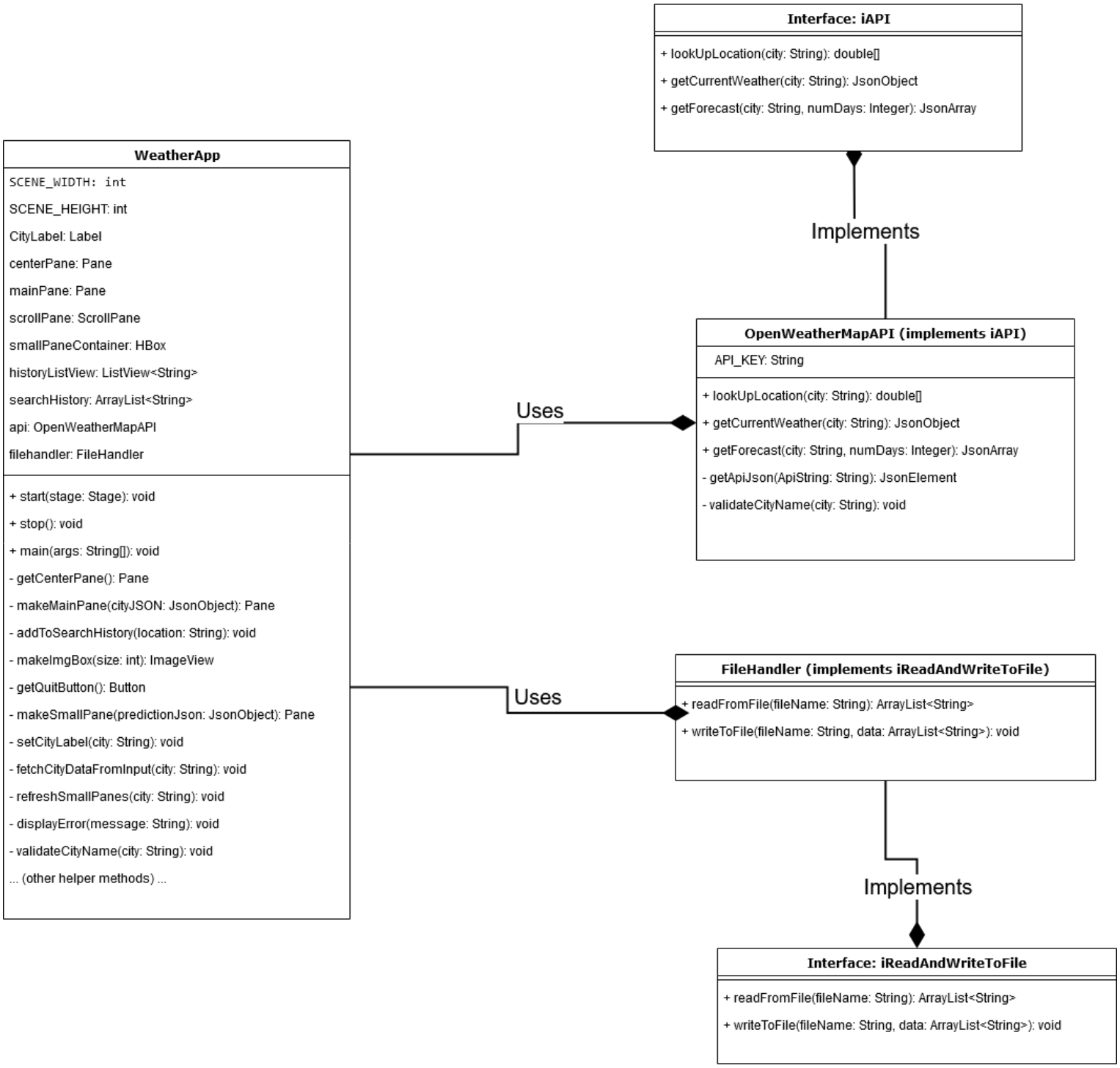
Interface: iAPI
+ lookUpLocation(city: String): double[]
+ getCurrentWeather(city: String): JsonObject
+ getForecast(city: String, numDays: Integer): JsonArray
Implements
WeatherApp
SCENE_WIDTH: int
SCENE_HEIGHT: int
CityLabel: Label
centerPane: Pane
mainPane: Pane
scrollPane: ScrollPane
smallPaneContainer: HBox
historyListView: ListView<String>
searchHistory: ArrayList<String>
api: OpenWeatherMapAPI
filehandler: FileHandler
+ start(stage: Stage): void
+ stop(): void
+ main(args: String[]): void
- getCenterPane(): Pane
- makeMainPane(cityJSON: JsonObject): Pane
- addToSearchHistory(location: String): void
- makeImgBox(size: int): ImageView
- getQuitButton(): Button
- makeSmallPane(predictionJson: JsonObject): Pane
- setCityLabel(city: String): void
- fetchCityDataFromInput(city: String): void
- refreshSmallPanes(city: String): void
- displayError(message: String): void
- validateCityName(city: String): void
... (other helper methods) ...
Uses
OpenWeatherMapAPI (implements iAPI)
API_KEY: String
+ lookUpLocation(city: String): double[]
+ getCurrentWeather(city: String): JsonObject
+ getForecast(city: String, numDays: Integer): JsonArray
- getApiJson(ApiString: String): JsonElement
- validateCityName(city: String): void
Uses
FileHandler (implements iReadAndWriteToFile)
+ readFromFile(fileName: String): ArrayList<String>
+ writeToFile(fileName: String, data: ArrayList<String>): void
Implements
Interface: iReadAndWriteToFile
+ readFromFile(fileName: String): ArrayList<String>
+ writeToFile(fileName: String, data: ArrayList<String>): void

Responsibilities of Key Classes:

1. Class: WeatherApp

    Responsibilities:
        User Interface Management: Create and manage the graphical user interface (GUI) elements like buttons, labels, text fields, etc.
        Event Handling: Respond to user actions such as button clicks.
        Application Flow Control: Manage the overall flow of the application, including starting and stopping the app.
        Data Display: Fetch and display weather data, error messages, and history.

2. Class: OpenWeatherMapAPI (Implements iAPI)

    Responsibilities:
        Data Retrieval: Connect to the OpenWeatherMap API to fetch weather data based on city names.
        Data Processing: Process the JSON responses from the API and extract necessary data.
        Input Validation: Validate inputs like city names and number of forecast days before making API calls.

3. Class: FileHandler (Implements iReadAndWriteToFile)

    Responsibilities:
        File Reading: Read data (like search history) from a file.
        File Writing: Write data (like search history) to a file.
        Data Conversion: Convert data between formats suitable for file storage and application use.

4. Interface: iAPI

    Responsibilities:
        API Contract Definition: Define the contract for weather data retrieval,
       including methods for looking up locations, getting current weather, and fetching forecasts.

5. Interface: iReadAndWriteToFile

    Responsibilities:
        File Operation Definition: Define the contract for file read and write operations,
       specifying methods for reading from and writing to files.

---

Project Functionality Documentation

WeatherApp is a JavaFX-based desktop application designed to provide real-time weather information and forecasts. Utilizing data from the OpenWeatherMap API, it offers users an intuitive interface to search for weather data by city names and view the current weather conditions as well as future forecasts.
Features and Capabilities
Weather Data Display

Current Weather:
    Displays the current weather information for a user-specified city. This includes temperature, wind speed, humidity, and a brief description of the current weather (e.g., sunny, cloudy)

    Graphical Representation: Weather conditions are accompanied by representative icons or images, enhancing the visual appeal and user experience.

Weather Forecast:

    Forecast Data: Provides a forecast for the specified city. The default setting displays weather predictions for the next few days, including temperature ranges and general weather conditions.

    Forecast Details: Each day's forecast includes specific details such as expected high and low temperatures, wind conditions, and the likelihood of precipitation.

Search Functionality:

    City Search: Users can search for weather information by entering the name of a city. The application validates the input to ensure it's a valid city name.

    Search History: The app maintains a history of recent searches, allowing users to quickly revisit previous queries.

User Interface:

    Main Window: The application's main window features a clean layout with a search bar, a display area for weather information, a history list of recent searches, and a quit button.

    Responsive Design: The UI dynamically adjusts to display the retrieved weather data effectively.

User Interaction and Workflow:

Starting the Application: Upon launch, the application presents a user interface with an empty state, ready for the first search.

Performing a Search: Users enter a city name in the search bar and initiate the search by pressing the 'Search' button.

Viewing Weather Data: Once a search is performed, current weather and forecast data for the entered city are displayed in the main window.

Navigating Search History: Users can view their recent searches in the history list and select any previous search to quickly view the weather data for that city again.

External Integrations:

OpenWeatherMap API: WeatherApp retrieves all weather-related data from the OpenWeatherMap API. This includes real-time weather conditions and forecasts.

Error Handling and Messages:

Invalid City Names: If a user enters a non-existent or incorrectly spelled city name, the application displays an error message indicating the issue.

Network Issues: In case of network problems or API unavailability, the application informs the user about the connectivity issue.

API Limitations: Any limitations or errors returned by the OpenWeatherMap API are communicated to the user through appropriate messages.

---

class preconditions:

For fetchCityDataFromInput(String city):
The city parameter must not be null or empty.
The city parameter must only contain alphabetic characters and spaces.

For getForecast(String city, Integer numDays):
The city parameter must not be null, empty, and must be a valid city name.
The numDays parameter must be a positive integer.

Class postconditions:

For fetchCityDataFromInput(String city):
If the city is valid and data is available, the weather data for the specified city is fetched and displayed.
If the city is invalid or data is unavailable, an error message is shown to the user.

For getForecast(String city, Integer numDays):
Returns a JsonArray containing the forecast data for the specified number of days.
If no data is available or if an error occurs, the returned JsonArray is empty or an error is handled.

---

Agreed division of work:
None

Actual division of work:
- Getting the API key and setting up the project - Juho Naatula
- converting the app into it's own UI layout based on Panes - Juho Naatula
- Designing how the UI should look like and how it should function - Juho Naatula
- Adding a way to make API calls to the OpenWeatherMap API - Juho Naatula
- Implementing the base for searching up locations and fetching their Json data within the UI - Juho Naatula
- Cleaning up the code by making support functions for unpacking the API Json data - Juho Naatula
- Coding the UI elements to show current weather - Juho Naatula
- Adding API calling methods and UI components for getting weather forecasts - Juho Naatula
- Getting weather icons from OpenWeatherMap and displaying them in the UI - Juho Naatula
- Adding a custom wind icon that rotates and displays the direction of the wind - juho Naatula
- Adding the basic code for recording and displaying search history - Olayinka Akande Gold
- Fixing the implementation of displaying search history and adding its functionality - Juho Naatula
- Moved and modified the code to use the Intefaces - Juho Naatula
- Adding comments and documentation to the code and Interfaces - Juho Naatula
- Adding error handling to the code and Interfaces - Juho Naatula
- Making the UML class responsibility charts - Juho Naatula

---

User Manual for WeatherApp
Introduction:

Welcome to WeatherApp, a user-friendly desktop application that provides real-time weather information and forecasts. This manual will guide you through the basic functionalities of the WeatherApp and how to use it effectively.
Getting Started

- Launching the Application:
  Package the maven project from the main directory using the command in the cmd:
  "mvn package"

  And then launch the program with the command in the cmd:
  "java -jar target\WeatherApp-1.0.one-jar.jar"

- Main Interface:
  Upon launching, you will see the main window divided into several sections: a search bar, a weather information display area, weather forecast, and a quit button.

How to Use WeatherApp
Performing a Weather Search:

- Enter City Name:
  Locate the search bar at the top of the application.
  Type the name of the city for which you want to check the weather.
  Ensure the city name is correctly spelled.

- Initiate Search:
  After entering the city name, click on the 'Search' button.
  The application will now retrieve weather data for the specified city.

Viewing Weather Information:

- Current Weather:
  Once the search is complete, the current weather details for the selected city will be displayed. This includes temperature, wind speed, and a general description of the weather condition.

- Weather Forecast:
  Below the current weather information, you will find the forecast for the next few days. Each day's forecast includes temperature ranges and general conditions.

Using Search History:

- Viewing History:
  The application automatically saves your recent searches. You can view these in the 'History' section.
  Click on any city name in the history list to quickly view the weather information for that location.

Exiting the Application:

- Quit Button:
  To exit the application, click the 'Quit' button located at the bottom of the window.

Error Handling

- Invalid City Names: If an invalid city name is entered, an error message will be displayed. Please check the spelling and try again.

- Connectivity Issues: If there are any problems with your internet connection or issues connecting to the weather service, an error message will inform you of these.

---

Known bugs:

- If there are duplicate city names, we just pick the first one so you might not get the correct city from the list of duplicates
- If a city has a local name and a English name, the local name won’t return a valid response from the API, example: København (Danish name, error) - Copenhagen (English name, fine)
- Failed to find weather data for some smaller cities

Missing features:

- No way to save favorite locations
- No detailed forecast, I applied for the extended API access with the student application, they never accepted it so i never gained access to it, so i wasnt able to implement it
- No unit testing