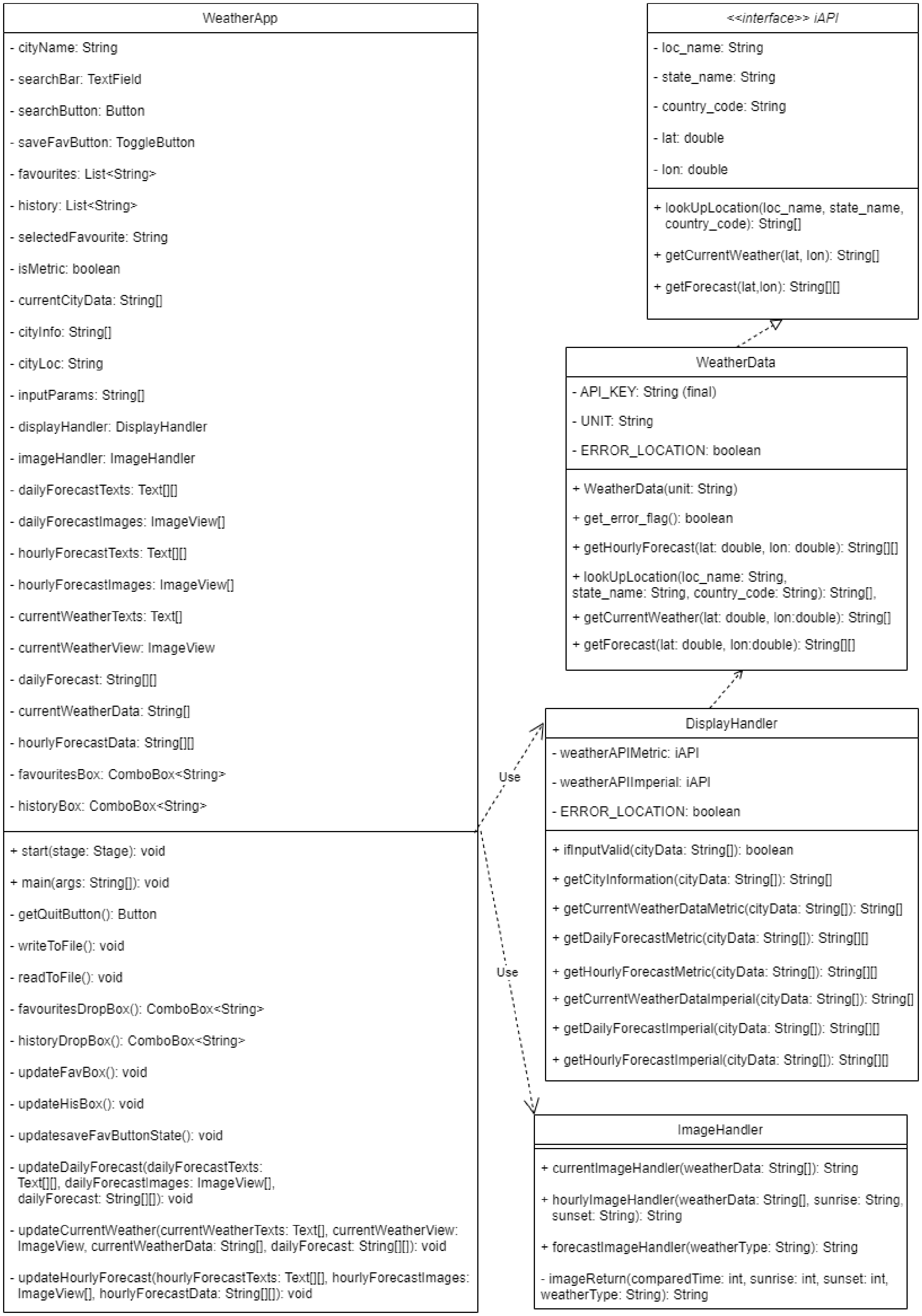
Project: Weather App

Minh Vu, Hieu Hoang, Minh Khanh Pham

I. Structure of the software


WeatherApp
- cityName: String
- searchBar: TextField
- searchButton: Button
- saveFavButton: ToggleButton
- favourites: List<String>
- history: List<String>
- selectedFavourite: String
- isMetric: boolean
- currentCityData: String[]
- cityInfo: String[]
- cityLoc: String
- inputParams: String[]
- displayHandler: DisplayHandler
- imageHandler: ImageHandler
- dailyForecastTexts: Text[][]
- dailyForecastImages: ImageView[]
- hourlyForecastTexts: Text[][]
- hourlyForecastImages: ImageView[]
- currentWeatherTexts: Text[]
- currentWeatherView: ImageView
- dailyForecast: String[][]
- currentWeatherData: String[]
- hourlyForecastData: String[][]
- favouritesBox: ComboBox<String>
- historyBox: ComboBox<String>
+ start(stage: Stage): void
+ main(args: String[]): void
- getQuitButton(): Button
- writeToFile(): void
- readToFile(): void
- favouritesDropBox(): ComboBox<String>
- historyDropBox(): ComboBox<String>
- updateFavBox(): void
- updateHisBox(): void
- updatesaveFavButtonState(): void
- updateDailyForecast(dailyForecastTexts: Text[][], dailyForecastImages: ImageView[], dailyForecast: String[][]): void
- updateCurrentWeather(currentWeatherTexts: Text[], currentWeatherView: ImageView, currentWeatherData: String[], dailyForecast: String[][]): void
- updateHourlyForecast(hourlyForecastTexts: Text[][], hourlyForecastImages: ImageView[], hourlyForecastData: String[][]): void
<<interface>> iAPI
- loc_name: String
- state_name: String
- country_code: String
- lat: double
- lon: double
+ lookUpLocation(loc_name, state_name, country_code): String[]
+ getCurrentWeather(lat, lon): String[]
+ getForecast(lat,lon): String[][]
WeatherData
- API_KEY: String (final)
- UNIT: String
- ERROR_LOCATION: boolean
+ WeatherData(unit: String)
+ get_error_flag(): boolean
+ getHourlyForecast(lat: double, lon: double): String[][]
+ lookUpLocation(loc_name: String, state_name: String, country_code: String): String[],
+ getCurrentWeather(lat: double, lon:double): String[]
+ getForecast(lat: double, lon:double): String[][]
Use
Use
DisplayHandler
- weatherAPIMetric: iAPI
- weatherAPIImperial: iAPI
- ERROR_LOCATION: boolean
+ ifInputValid(cityData: String[]): boolean
+ getCityInformation(cityData: String[]): String[]
+ getCurrentWeatherDataMetric(cityData: String[]): String[]
+ getDailyForecastMetric(cityData: String[]): String[][]
+ getHourlyForecastMetric(cityData: String[]): String[][]
+ getCurrentWeatherDataImperial(cityData: String[]): String[]
+ getDailyForecastImperial(cityData: String[]): String[][]
+ getHourlyForecastImperial(cityData: String[]): String[][]
ImageHandler
+ currentImageHandler(weatherData: String[]): String
+ hourlyImageHandler(weatherData: String[], sunrise: String, sunset: String): String
+ forecastImageHandler(weatherType: String): String
- imageReturn(comparedTime: int, sunrise: int, sunset: int, weatherType: String): String

II. Responsibilities of the key classes

- The class “WeatherData”, which implements the “iAPI” interface, serves as the back-end as it performs the API calls to OpenWeatherMap to retrieve weather information, and some other related data, to show on the GUI.
- The class “DisplayHandler” oversees managing the weather data gathered from the user input and gives the UI appropriate data sets to show to the user.
- The class “ImageHandler” specifically handles the resources of the program. It manages different images by showing them in appropriate weather types such as cloudy, rain, etc. It uses the weather data from the user input and returns specific images.
- The class “WeatherApp” is the front-end class of the program. It creates the UI for the user. It collects the user inputs and displays weather data based on those inputs. It shows the current weather, a 4-day forecast and an hourly forecast.

III. Project functionality

The program serves as a conventional weather app, where you can look for the current weather information, weather forecasts for the next three days, and hourly forecasts for the next 24 hours of mostly every location on Earth. The weather information includes, for example, average temperature, highest and lowest points, wind speed. Here are some features of the program, which are listed below:

- Weather information can be displayed in either metric unit or imperial unit.
- Favourite locations and the current location are stored in files when the program closes and read again when the program starts. When users choose the location from the drop boxes, the program will execute a search command to show the weather information of that location.
- Users can also save and unsave a location as their favourites
- The state of the program will remain unchanged after closing using Quit button

Additional features are:

- Unit tests have been implemented using continuous integration
- Weather information can be displayed in either metric unit or imperial unit.
- Location search history. Recently searched locations are kept in memory and displayed to the user to make it easier to go back to them later. Search history is also saved in a file and restored on program restart.

Own features are:

- The users can even deduce both the current local time and the sunrise/sunset time of a location by examining the hourly forecast. This is because each timestamp in this section is marked with the local hour, and the weather icon corresponding to each time indicates either a sun or a moon.
- Users can clear search history.

IV. User manual

- Like a standard weather app, users can input a location's name into the search bar and press "Search" button to find its weather information. They can specify the country by typing its ISO 3166 country code after the location's name, separated by a comma. If the user wants to specify the state of the searching location (for US locations), they must type in the location's name, state's name, and even the country code, separated by comma ",".
- The "Change Unit" button changes the unit from metric to imperial and vice versa, so the user can use this button to display whatever unit they want. When opening the program, the displayed unit will be the displayed unit when it was stopped.
- "Save as favourite" button saves the location in "Favourite" drop box, then the button will change to "Unsave", and the users can unsave the location from the drop box.
- Users can choose the location from the "History" and "Favourite" drop boxes.
- Users can use "Clear History" to clear all searched locations
- Users should use "Quit" button to close the program so that all the data is stored

V. The agreed and actual division of the work

	Minh Vu	Hieu Hoang	Minh Khanh Pham
Agreed	Implementing the class "WeatherData" that serves as the back end, which performs the API calls to OpenWeatherMap	- Implement the UI of the program – the class "WeatherApp" - Search for an icon library	- Implement the functionalities and the buttons of saving favorite, and search locations - Draw UML diagram of the program
Actual	- Implementing the class "WeatherData" of the program. - Implement the unit tests for the class files. - Manage group meetings and group's progress.	- Implement the UI for the program - Refactor the codes - Give feedback to groupmates	- Implement the functionalities and the buttons of saving favorite, and search locations - Draw UML diagram of the program - Refactor the codes - Give feedback to groupmates

VI. Known bugs

The state specification only works for US locations, following how the Geocoding API call works, so if the user tries to specify state for non-US locations, then in some cases the error message "Location not found!" will pop up (but not in every case).