Project: Weather Forecast Data System

Overview

You are going to analyze a weather forecast system and create a class diagram based on how it works. This system pulls weather data from an online source, organizes it, and saves it in a structured format. It provides both **daily** and **hourly** weather reports, which can be accessed separately. A background process handles data collection so the system runs smoothly.

Representing External Components

External Systems in the Diagram

- **Weather API**: Provides weather forecast data based on a location's latitude and longitude. It receives requests from the system and returns structured weather data.
- **External Weather Service**: The online system that hosts and supplies the weather forecast data. This service powers the **Weather API**, meaning all data ultimately comes from this source.

These external systems should not be shown as classes but should be represented as **external dependencies** in your class diagram. You can represent these components using either a **package symbol** or a **plain rectangle** labeled appropriately to indicate that the system retrieves data from them, but they are not actual classes within the system.

The **Weather API** acts as an intermediary between the system and the **External Weather Service**. The system does not communicate directly with the external weather service but instead sends requests to the API, which fetches the necessary data from the service before returning it.

The **ForecastWorker** is responsible for making API calls and should be the connection between the system and the Weather API.

How the System Works

This system is made up of several key components that work together to retrieve, store, and display weather forecast data.

1. Retrieving Weather Data

- The system connects to an **external weather service** using an API.
- The API provides weather data for a specific location, including temperature, chance of rain, wind speed, and other relevant conditions.
- The system requests **two types of forecasts**: a **daily forecast** (covering multiple days) and an **hourly forecast** (covering a more detailed breakdown).
- This data is retrieved by a **ForecastWorker**, a background process that ensures the system remains responsive while waiting for data from the online service.

2. Storing and Organizing Data

- The retrieved forecast data is **saved in CSV files** for easier access and processing.
- The system has two main classes for storing forecasts:

    - **DailyForecast**: Stores forecast details for a given day.
    - **HourlyForecast**: Stores forecast details for a specific hour.
- Each forecast object stores **temperature values in both Fahrenheit and Celsius**, converting between the two when necessary.
- Forecast objects also include details like weather condition icons (e.g., ☀️ for sunny, 🌧️ for rain), precipitation probability, and wind information.

3. Managing Forecast Data

- The system uses two management classes:
    - **DailyForecastManager**: Loads and organizes daily forecast data.
    - **HourlyForecastManager**: Loads and organizes hourly forecast data.
- These managers read data from the CSV files and transform it into structured objects that can be used by the rest of the system.
- The system ensures data is formatted in a user-friendly way, including converting temperature units and formatting times.

Your Task

Your job is to **study the system** based on this description and figure out how it is structured. You will create a **class diagram** showing:

- The **main classes** and what information they store.
- How the classes **connect** to each other (e.g., which classes work together or depend on each other).
- What **functions or actions** each class might have.
- How the system processes and organizes forecast data.

Hints for Your Diagram

- Think about how **DailyForecast** and **HourlyForecast** are separate but related.
- Consider how the **ForecastWorker** fetches data and passes it to the managers.
- Look at how forecast data is stored in CSV files and then loaded into objects.
- Identify **which classes handle data formatting** (e.g., temperature conversion, emoji representation for icons).
- Show the **API and External Weather Service as external components** using either a **package symbol** or a **plain rectangle** in your diagram.

Your class diagram should not just be a direct copy of the system. Instead, it should be an **organized model** that represents how the system is structured and how different parts work together.

What to Submit

- Push your **class diagram** that visually represents the system's structure to the GitHub repo used for Concept Initiates 1 and 2.