Weather Forecast Application

Problem Statement

This program retrieves and displays weather forecasts from the **National Weather Service API** based on a user-provided location. It converts the location into geographic coordinates and fetches forecast data, including an **hourly breakdown** of key weather details.

Users confirm their location before the forecast is retrieved. The results are displayed in the terminal in a structured format and saved to a **CSV file**. The system includes error handling for invalid locations, API failures, and missing data.

Forecasts are retrieved only when requested. The program does not retain data beyond storing it in a file, and users must manually enter a location for each request. The format of the forecast is fixed, and there are no built-in options for adjusting what information is shown or how it is presented.

Concept Initiate 1

Project Directions

This is a professional assignment that should be taken seriously. Your work is expected to meet the standards outlined in lecture, including **clear, structured writing and proper formatting**. Your final submission should be **well-organized and free of errors**, demonstrating a thoughtful and thorough approach.

Your task is to develop **user stories** that describe how different people might interact with this system. Consider the **variety of users and needs** that exist beyond simply retrieving a forecast. Your user stories should reflect **different perspectives and use cases**, not just minor variations of the same idea.

1. **Identify different types of users.** What kinds of people might interact with this system? Consider how their needs, priorities, and expectations could vary.
2. **Think through the interaction process.** What steps are necessary to get weather information? Does the system provide everything a user might need? What challenges might someone face?
3. **Consider how forecast data is used.** The program saves data but does not offer built-in ways to access or analyze past forecasts. Is this enough? Are there alternative ways someone might want to use the information?

You are **encouraged to talk to others**—getting input from different people is one of the best ways to develop strong user stories. There is no strict requirement on the number of user stories, users, or features, but your submission must be **robust**. Your work should show variety in both the types of users represented and the functionality they might expect.

User Story Format

Each user story should include a **title** and follow this structure:

Title: (Brief description of the scenario)
As a [user], I want [goal] so that [benefit].

All work must be **pushed to your GitHub repository** in the lab section content available in **iLearn**.

Concept Initiate 2

Project Directions

Your task is to create a Use Case Diagram for the Weather Forecast Application. This diagram should visually represent the interactions between users and the system based on the user stories you developed in Concept Initiate 1. If you have an insuffient amount of stories or something else wrong, you will be docked points on both Concept Initiates. So, you are encouraged to recieve feedback for your Concept Initiate 1 solution before finalizing work on your Concept Initiate 2 solution. Each of your user stories should be represented in the diagram referenced by the title you gave them, showing how different types of users interact with the system.

Your diagram must be structured correctly using proper UML notation. Each component must use the correct shape to receive credit—***the use of incorrect symbols will result in the loss of all points for that category of the diagram***. Be sure to use appropriate actor icons, ovals for use cases, lines for associations, and a system boundary box around the use cases.

Your final submission must be exported as a PDF (.pdf) or image file (.png, .jpg, or .jpeg). Project files from diagramming tools (e.g., .drawio, .vsdx, .lucidchart) will not be accepted. Ensure that your file is in the correct format before submission. Hand-drawn diagrams will not be accepted. Your final submission should be exported as a PDF or image file and pushed to GitHub.

The diagram should be clear and easy to read. Elements should be spaced out as to minimize clutter and overlapping. The structure should make the relationships between actors and use cases easy to follow at a glance.

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.

Iteration 1

Completing the ForecastWorker

This phase focuses on completing the code for the `ForecastWorker` component of the Weather Forecast Application. This class is responsible for retrieving weather forecast data from the National Weather Service API. The partial implementation can be found in the `weather_app` directory in the file named `forecast_worker.py`.

The code you are expected to complete primarily focuses on interacting with **CSV files**—specifically, processing and saving daily and hourly forecast data into structured files. Each TODO (explained below) corresponds to a step in this process and supports the system's overall functionality for organizing and storing forecast data.

What Is a TODO?

A **TODO** is a comment used by developers to mark a place in the code where work needs to be done. These comments usually serve as placeholders or reminders to return to a specific section and complete it later. In Python, they typically look like this:

```
# TODO: Implement error handling for invalid responses
```

TODOs are ignored by the Python interpreter but can be picked up by development tools to help track outstanding tasks.

Using the TODO Tool Window in PyCharm

PyCharm automatically detects TODO comments and displays them in a centralized tool window for easy navigation.

To access the TODO tool window:

1. Go to the **bottom panel** of the PyCharm interface.
2. Click on the **"TODO"** tab.
    - If it's not visible, open it via the menu: `View > Tool Windows > TODO`

This window will show a list of all TODO comments in the project, grouped by file. Clicking any item in the list will take you directly to that line in the code.

This feature is especially helpful for locating unfinished code across multiple files without manually scrolling through everything.

Your Task

Your job in Iteration 1 is to:

- Open the file `forecast_worker.py` inside the `weather_app` directory.
- Locate all TODO comments in the file. You may use the TODO tool window in PyCharm to assist you.
- Carefully review the surrounding code and complete all the tasks indicated by the TODOs.
- Follow proper Python syntax and best practices.
- Keep your code readable and well-organized. Hint: repurpose the instructional comments to anotate your code.
- You can test your implementation by running the `main()` function at the bottom of the file. If successful, the weather forecast should be retrieved and written to the appropriate CSV file.

What to Submit

- Push your updated `forecast_worker.py` file to your GitHub repository.
- Make sure your changes are committed with clear and descriptive messages.
- Verify that all TODO comments have been resolved and removed unless explicitly marked to remain.

```
# Example commit message
git commit -m "Complete ForecastWorker implementation for Iteration 1"
```

Now feel proud of yourself even if you aren't proud of your work. You just completed the first step of your first major project for Computer Science.