Weather Application SRS

1. Basic Structure:
   * Create a main class WeatherApp.java as the entry point of your application.
   * Define a class TerminalUI.java to handle the terminal-based user interface.
2. Add Locations:
   * Implement a method in TerminalUI.java to allow users to add locations.
   * Prompt the user to input the latitude and longitude or city/country name.
   * Validate the input and store the locations in memory or a text file.
3. View Current Weather:
   * Implement a method to display the current weather conditions for a chosen location.
   * Retrieve weather data from the OpenWeatherMap API using the provided latitude and longitude or city/country name.
   * Parse the JSON response to extract relevant weather information such as temperature, humidity, etc.
   * Display the weather information to the user in the terminal.
4. View Basic Information:
   * Extend the current weather display method to include additional basic information such as feels like temperature, minimum and maximum temperature, etc.
5. View Sunrise and Sunset Time:
   * Modify the weather display method to include sunrise and sunset time for the chosen location.
6. View Weather Forecast for 5 Days:
   * Implement a method to fetch and display the weather forecast for the next 5 days with a 3-hour interval.
   * Parse the JSON response from the OpenWeatherMap API to extract forecast data.
   * Display the forecast information to the user in the terminal.
7. Add Timestamp for Weather Records:
   * Update the data storage mechanism to include timestamps for weather records.
   * Associate each weather record with the date and time it was fetched.
8. Cache Management:
   * Implement caching to store frequently accessed weather data in memory or a text file.
   * Check the cache before making API requests to minimize the number of API calls.
9. Generate Notification for Poor Weather Conditions:
   * Define thresholds for poor weather conditions (e.g., low temperature, high humidity).
   * Generate notifications when weather conditions meet or exceed these thresholds.
10. Air Pollution Data:
    * Extend the weather display method to include air pollution data for the chosen location.
    * Fetch air pollution data from the OpenWeatherMap API and display it to the user.
11. Generate Notification for Poor Air Quality:
    * Define thresholds for poor air quality based on air pollution data.
    * Generate notifications when air quality levels exceed these thresholds.
12. Polluting Gases Data:
    * Fetch and display information about polluting gases such as Carbon monoxide (CO), Nitrogen dioxide (NO2), etc., for the chosen location.

Functions For TerminalUI.java:

1. displayMenu(): Display the main menu options to the user.
2. addLocationByCoordinates(): Prompt the user to input latitude and longitude coordinates to add a location for weather checking.
3. addLocationByCityName(): Prompt the user to input a city name to add a location for weather checking.
4. getCurrentWeather(): Retrieve and display the current weather conditions for a selected location.
5. displayBasicWeatherInfo(): Display basic weather information like temperature, humidity, wind speed, etc.
6. displaySunriseSunsetTime(): Display sunrise and sunset times for a selected location.
7. displayWeatherForecast(): Display the weather forecast for the next 5 days.
8. addTimestampForWeatherRecords(): Add a timestamp for weather records to track when the data was last updated.
9. implementCacheManagement(): Implement caching to store frequently accessed weather data and improve application performance.
10. generateWeatherNotifications(): Generate notifications for poor weather conditions to alert the user.
11. displayAirPollutionData(): Display air pollution data including Air Quality Index and information about polluting gases.
12. generateAirQualityNotifications(): Generate notifications for poor air quality to alert the user.

Components (JAVA SWING) and Functions For Desktop UI:

1. displayMenu ():(Optional)
   * Purpose: Display the main menu options to the user.
   * UI Component: Menu bar or side navigation panel.
2. addLocationByCoordinates():
   * Purpose: Allow users to add locations by entering latitude and longitude coordinates.
   * UI Component: Text fields for latitude and longitude input.
3. addLocationByCityName():
   * Purpose: Allow users to add locations by entering city names.
   * UI Component: Text field for city name input.
4. getCurrentWeather():
   * Purpose: Retrieve and display the current weather conditions for a selected location.
   * UI Component: Weather information panel or card.
5. displayBasicWeatherInfo():
   * Purpose: Display basic weather information like temperature, humidity, wind speed, etc.
   * UI Component: Labels or text fields within the weather information panel.
6. displaySunriseSunsetTime():
   * Purpose: Display sunrise and sunset times for a selected location.
   * UI Component: Labels or text fields within the weather information panel.
7. displayWeatherForecast():
   * Purpose: Display the weather forecast for the next 5 days.
   * UI Component: Forecast panel or table with multiple rows for each day's forecast.
8. addTimestampForWeatherRecords():
   * Purpose: Add a timestamp for weather records to track when the data was last updated.
   * UI Component: Text field or label to display the timestamp.
9. implementCacheManagement():
   * Purpose: Implement caching to store frequently accessed weather data and improve application performance.
   * UI Component: N/A (Backend functionality).
10. generateWeatherNotifications():
    * Purpose: Generate notifications for poor weather conditions to alert the user.
    * UI Component: Notification pop-up or alert dialog.
11. displayAirPollutionData():
    * Purpose: Display air pollution data including Air Quality Index and information about polluting gases.
    * UI Component: Air quality information panel or card.
12. generateAirQualityNotifications():
    * Purpose: Generate notifications for poor air quality to alert the user.
    * UI Component: Notification pop-up or alert dialog

FULL PATHWAY:

weather-application/

│

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ ├── ui/

│ │ │ │ ├── TerminalUI.java // Terminal-based UI implementation

│ │ │ │ └── DesktopUI.java // Desktop application UI implementation

│ │ │ │

│ │ │ ├── storage/

│ │ │ │ ├── SQLStorage.java // SQL-based storage implementation

│ │ │ │ └── TextFileStorage.java // Text file-based storage implementation

│ │ │ │

│ │ │ └── service/ // Service or business layer

│ │ │ ├── WeatherService.java

│ │ │ ├── AirQualityService.java

│ │ │ └── CacheService.java

│ │ │

│ │ └── resources/ // Configuration files, API keys, etc.

│ │

│ └── test/ // Unit tests

│ └── java/

│ ├── ui/

│ ├── storage/

│ └── service/

│

└── pom.xml // Maven project configuration