

CSE-214 Online - 2 (A1)

Structural Design Pattern

Suppose you're working on a weather forecasting application that consumes data from multiple third-party weather APIs. One of the third-party services provides weather data in a specific format through a `LegacyWeatherService` class, which has a method called `getWeatherData`. However, your application expects data to be accessed through a standardized `WeatherProvider` interface with a `fetchWeather` method.

You cannot modify the `LegacyWeatherService` class because it's a closed-source library.

Provide a solution with an appropriate design pattern that allows the application to access the legacy weather data through the `WeatherProvider` interface, ensuring compatibility without altering the legacy code.

```
// Legacy Weather Service, which we cannot modify
class LegacyWeatherService {
    public String getWeatherData() {
        return "Legacy weather data";
    }
}

// Client Interface expected by the application
interface WeatherProvider {
    String fetchWeather();
}

// Application class that depends on the WeatherProvider interface
class WeatherApp {
    private WeatherProvider weatherProvider;

    public WeatherApp(WeatherProvider weatherProvider) {
        this.weatherProvider = weatherProvider;
    }

    public void displayWeather() {
        System.out.println(weatherProvider.fetchWeather());
    }
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Legacy service instance  
        LegacyWeatherService legacyWeatherService = new LegacyWeatherService();  
  
        ??  
  
        WeatherApp app = ??  
        app.displayWeather(); // Output: Legacy weather data  
    }  
}
```