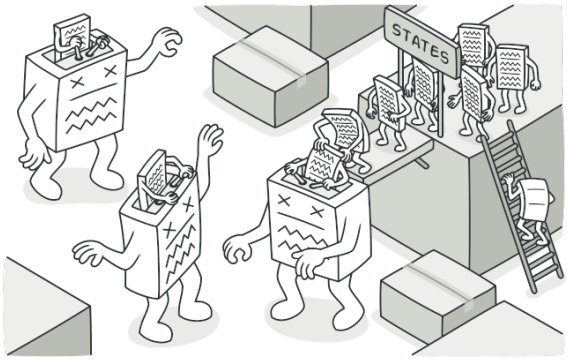
Design Patterns

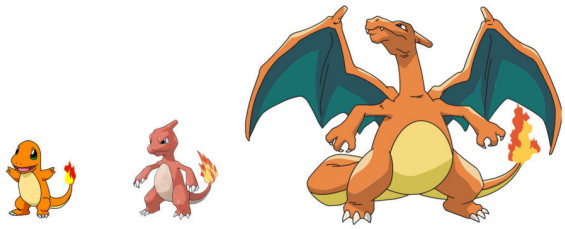
Edwin Carlinet

State



State is a a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

Problem



Suppose that you want to model the Pokemon game. A Pokemon is a monster that can do many things like attack, defend, heal, and **upgrade**.

Obviously, the Pokemon's behavior changes depending on its "type". For example, when after evolving, a Pokemon can learn new moves, change its appearance, and increase its stats.

How do you model the Pokemon's behavior in a way that allows it to change its behavior when it evolves?

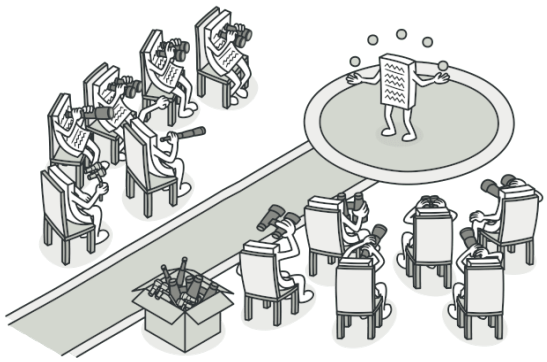
Exercise

A turn is composed of four steps: attack, defend, heal, and upgrade. The fight function takes two Pokemons and makes them fight until one of them is defeated.

```
def fight(a: Pokemon, b: Pokemon):
    while a.is_alive():
        attack = a.attack(b)
```

```
b.defend(attack)
b.heal()
a.upgrade() # May or may not upgrade
a, b = b, a
```

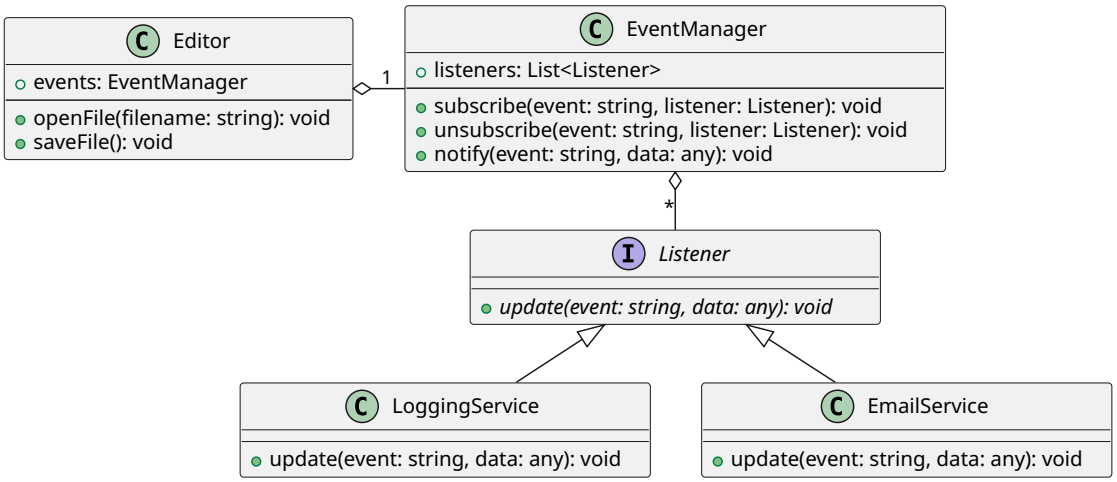
Observer



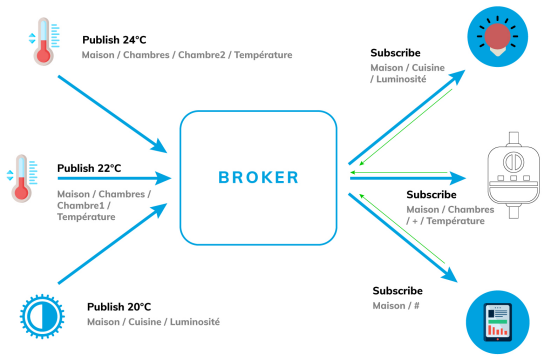
Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

Example

In this example, the Observer pattern lets the text editor object notify other service objects about changes in its state.



Exercise



You are developing an IoT application that monitors temperature sensors deployed in various locations. The application uses the MQTT (Message Queuing Telemetry Transport) protocol for real-time data exchange between sensors and the central server. Your task is to implement a system that notifies subscribers when the temperature exceeds a certain threshold.

You have to implement the MQTT broker !!!

1. **Temperature Sensors** Simulate temperature sensors that periodically publish temperature readings to MQTT topics.
2. **MQTT Broker** Set up an MQTT broker to facilitate communication between sensors and subscribers.
3. **Subscribers** Implement some subscribers that listens to MQTT topics for temperature data:
   - *Alarm*: triggers alerts when the temperature exceeds a predefined threshold.
   - *Phone App*: displays an alert on a mobile app.
   - *Climate Control System*: adjusts the air conditioning based on the temperature readings.

!!! note A broker have topics (e.g. “sensors/temperature”), you push notifications in a topic.

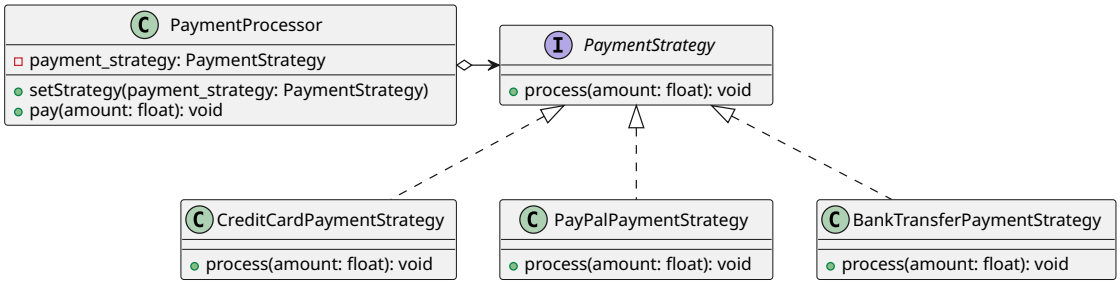
You have to use the *Observer Design Pattern* to allow subscribers to register and receive notifications about temperature readings from sensors.

Strategy design pattern

Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

Example 💵

You are developing a payment processing system for an e-commerce platform. The system needs to support multiple payment methods, such as credit card, PayPal, and bank transfer. Your task is to implement a flexible and extensible system that allows users to choose their preferred payment method at checkout.



Exercise

You are implementing an AI system to play chess. The AI should be able to use different strategies to evaluate the best move to play. You have to implement the Strategy design pattern to be able to compare strategies and allow the AI to switch between different strategies at runtime. Strategies can be:

- Random: the AI plays a random move.
- MinMax: the AI plays the move that maximizes its chances of winning.