Il pattern **Observer/Observable** è un modello di progettazione che consente a un oggetto (detto **Observable**) di notificare automaticamente una serie di oggetti **Observer** quando il suo stato cambia. Questo pattern è molto utile quando si desidera mantenere diversi componenti sincronizzati senza creare un forte accoppiamento tra di essi.

Componenti del Pattern Observer/Observable

1. **Observable**: L'oggetto che contiene lo stato e notifica i cambiamenti ai suoi osservatori.
2. **Observer**: L'oggetto che si registra per ricevere le notifiche dall'Observable e aggiorna il suo stato di conseguenza.

Come Funziona

- L'Observable mantiene una lista degli Observer registrati.
- Quando lo stato dell'Observable cambia, notifica tutti gli Observer registrati.
- Gli Observer implementano un metodo specifico (di solito `**update**`), che viene chiamato dall'Observable per notificare il cambiamento.

Esempio con Pokémon

Vediamo come implementare il pattern Observer/Observable in un contesto Pokémon. In questo esempio, un Pokémon (Observable) può aumentare di livello, e un'interfaccia grafica (Observer) viene aggiornata automaticamente quando il livello del Pokémon cambia.

Modello (Observable)

Pokémon

java — Copia codice

```java
import java.util.ArrayList;
import java.util.List;

public class Pokemon {
    private String nome;
    private int livello;
    private List<PokemonObserver> observers = new ArrayList<>();

    public Pokemon(String nome, int livello) {
        this.nome = nome;
        this.livello = livello;
    }

    public String getNome() {
        return nome;
    }

    public int getLivello() {
        return livello;
    }

    public void setLivello(int livello) {
        this.livello = livello;
        notifyObservers();
    }

    public void addObserver(PokemonObserver observer) {
        observers.add(observer);
    }

    public void removeObserver(PokemonObserver observer) {
        observers.remove(observer);
    }

    private void notifyObservers() {
        for (PokemonObserver observer : observers) {
            observer.update(livello);
        }
    }
}
```

Observer

Interfaccia Observer

java — Copia codice

```java
public interface PokemonObserver {
    void update(int nuovoLivello);
}
```

Vista (Observer)

java — Copia codice

```java
public class PokemonView implements PokemonObserver {
    @Override
    public void update(int nuovoLivello) {
        System.out.println("Il Pokémon è salito di livello! Nuovo livello: " + nuovoLivello);
    }

    public void showDetails(String nome, int livello) {
        System.out.println("Nome: " + nome + ", Livello: " + livello);
    }
}
```

Controller

java — Copia codice

```java
public class PokemonController {
    private Pokemon model;
    private PokemonView view;

    public PokemonController(Pokemon model, PokemonView view) {
        this.model = model;
        this.view = view;
        this.model.addObserver(view);
    }

    public void setLivelloPokemon(int livello) {
        model.setLivello(livello);
    }

    public void showPokemon() {
        view.showDetails(model.getNome(), model.getLivello());
    }
}
```

Applicazione Principale

java — Copia codice

```java
public class MainApp {
    public static void main(String[] args) {
        Pokemon pikachu = new Pokemon("Pikachu", 5);
        PokemonView view = new PokemonView();
        PokemonController controller = new PokemonController(pikachu, view);

        controller.showPokemon(); // Mostra i dettagli iniziali

        // Simula il salire di livello del Pokémon
        controller.setLivelloPokemon(10);
        controller.setLivelloPokemon(15);
    }
}
```

Come Funziona Questo Esempio

1. **Observable (Pokemon)**: Il Pokémon è l'oggetto osservabile. Tiene traccia degli osservatori (oggetti `**PokemonObserver**`) e li notifica quando il livello cambia.
2. **Observer (PokemonView)**: La vista si registra come osservatore del Pokémon. Implementa il metodo `**update**`, che viene chiamato dall'Observable quando il livello cambia.
3. **Controller**: Il controller gestisce l'interazione tra il modello (Pokemon) e la vista (PokemonView). Quando il livello del Pokémon viene aggiornato tramite il controller, il modello notifica automaticamente la vista del cambiamento.

Benefici del Pattern Observer/Observable

- **Decoupling:** Gli Observer non hanno bisogno di conoscere i dettagli dell'Observable, solo che esso esiste e che possono registrarsi per essere notificati.
- **Manutenzione:** Separare la logica di aggiornamento dei dati dalla logica di presentazione rende il codice più facile da mantenere e modificare.
- **Estensibilità:** È facile aggiungere nuovi tipi di Observer senza modificare l'Observable.