Il **Chain of Responsibility** è un pattern comportamentale che permette di passare una richiesta lungo una catena di handler. Ogni handler può elaborare la richiesta o passarla al successivo nella catena.

Componenti del Chain of Responsibility

1. **Handler (Gestore)**: Un'interfaccia che dichiara un metodo per gestire le richieste.
2. **ConcreteHandler (Gestore Concreto)**: Implementa il metodo di gestione e può passare la richiesta al prossimo handler.
3. **Client (Cliente)**: Invia richieste a un handler della catena.

Esempio con Pokémon

Immagina di voler gestire una sequenza di eventi durante una battaglia, come controllare se un attacco colpisce, calcolare il danno, applicare effetti di stato.

Interfaccia Handler

java Copia codice

```java
public abstract class Handler {
    protected Handler next;

    public void setNext(Handler next) {
        this.next = next;
    }

    public abstract void handleRequest(Pokemon attaccante, Pokemon difensore);
}
```

ConcreteHandler

java Copia codice

```java
public abstract class Handler {
    protected Handler next;

    public void setNext(Handler next) {
        this.next = next;
    }

    public abstract void handleRequest(Pokemon attaccante, Pokemon difensore);
}
```

ConcreteHandler

java Copia codice

```java
public class CheckAccuracyHandler extends Handler {
    @Override
    public void handleRequest(Pokemon attaccante, Pokemon difensore) {
        System.out.println("Controllo dell'accuratezza dell'attacco...");
        if (next != null) {
            next.handleRequest(attaccante, difensore);
        }
    }
}

public class CalculateDamageHandler extends Handler {
    @Override
    public void handleRequest(Pokemon attaccante, Pokemon difensore) {
        System.out.println("Calcolo del danno...");
        if (next != null) {
            next.handleRequest(attaccante, difensore);
        }
    }
}

public class ApplyStatusEffectHandler extends Handler {
    @Override
    public void handleRequest(Pokemon attaccante, Pokemon difensore) {
        System.out.println("Applicazione dell'effetto di stato...");
        if (next != null) {
            next.handleRequest(attaccante, difensore);
        }
    }
}
```

Applicazione Principale

java Copia codice

```java
public class MainApp {
    public static void main(String[] args) {
        Handler accuracyHandler = new CheckAccuracyHandler();
        Handler damageHandler = new CalculateDamageHandler();
        Handler statusEffectHandler = new ApplyStatusEffectHandler();

        accuracyHandler.setNext(damageHandler);
        damageHandler.setNext(statusEffectHandler);

        Pokemon pikachu = new Pokemon("Pikachu");
        Pokemon charmander = new Pokemon("Charmander");

        accuracyHandler.handleRequest(pikachu, charmander);
        // Output:
        // Controllo dell'accuratezza dell'attacco...
        // Calcolo del danno...
        // Applicazione dell'effetto di stato...
    }
}
```