Riassunti behavioral patterns

Template Method (TM)

- Intento: Definire una struttura per un algoritmo, lasciando che le sottoclassi ne implementino alcuni passi. In questo modo, è possibile ridefinire solo alcuni passi di un algoritmo senza dover riscrivere l'intero algoritmo.
- Partecipanti:
  - **Classe astratta**: definisce l'interfaccia per l'algoritmo.
  - **Classe concreta**: implementa i passi dell'algoritmo che devono essere ridefiniti.
- Collaborazioni: la classe astratta collabora con le classi concrete per implementare l'algoritmo.
- Conseguenze:
  - Permette di ridefinire solo alcuni passi di un algoritmo senza dover riscrivere l'intero algoritmo.
    - Promuove l'incapsulamento delle classi concrete.
  - Bisogna capire quali passi dell'algoritmo devono essere ridefiniti.
- Pattern correlati:
  - **Factory Method**: può essere usato per creare un oggetto che implementa un passo dell'algoritmo.
  - **Strategy**: può essere usato per implementare un passo dell'algoritmo.

Interpreter (IN)

- Intento: Definire una grammatica per un linguaggio e implementare un interprete per tale linguaggio.
- Partecipanti:
  - **Espressione**: definisce un'interfaccia per l'interpretazione di un contesto.
  - **Espressione terminale**: implementa un'operazione terminale.
  - **Espressione non terminale**: implementa un'operazione non terminale.
- Collaborazioni: le espressioni terminali e non terminali collaborano per interpretare un contesto.
- Conseguenze:
  - Permette di definire un linguaggio per rappresentare un problema e implementare un interprete per tale linguaggio.
  - Facilita l'aggiunta di nuove regole grammaticali.
    - Grammatiche complesse possono essere difficili da gestire.
- Pattern correlati: Composite, Iterator, Visitor, Flyweight.

Mediator (ME)

- Intento: Definire un oggetto che incapsula le interazioni tra un gruppo di oggetti. Promuove il disaccoppiamento tra gli oggetti. Evita che gli oggetti si riferiscano esplicitamente gli uni agli altri.
- Partecipanti:
  - **Mediator**: definisce un'interfaccia per la comunicazione tra i colleghi.
  - **Collega**: definisce un'interfaccia per la comunicazione con il mediatore.
  - **Collega concreto**: implementa un collega.
  - **Mediatore concreto**: implementa un mediatore.
- Collaborazioni: i colleghi comunicano tra loro attraverso il mediatore.
- Conseguenze:
  - Promuove il disaccoppiamento tra gli oggetti.
  - Evita che gli oggetti si riferiscano esplicitamente gli uni agli altri.
  - Controllo centralizzato delle interazioni tra gli oggetti.
  - Limita la sottoclassificazione.
- Pattern correlati: Observer, Facade, Singleton.

Chain of Responsibility (CoR)

- Intento: Evitare di accoppiare il mittente di una richiesta al suo destinatario, dando a più oggetti la possibilità di gestire la richiesta. Collegare gli oggetti riceventi in una catena e passare la richiesta lungo la catena fino a quando un oggetto non la gestisce.
- Partecipanti:
  - **Handler**: definisce un'interfaccia per la gestione delle richieste.
  - **Handler concreto**: implementa un handler.
  - **Client**: inoltra le richieste agli handler.
- Collaborazioni: il client inoltra le richieste agli handler. Gli handler possono decidere di gestire la richiesta o di passarla al successivo handler nella catena.
- Conseguenze:
  - **Vantaggi**: evita di accoppiare il mittente di una richiesta al suo destinatario, dando a più oggetti la possibilità di gestire la richiesta.
  - **Svantaggi**: Accoppiamento ridotto. La richiesta può non essere gestita.
- Pattern correlati: Composite.

Mememto (MMT)

- Intento: Catturare e esternalizzare lo stato interno di un oggetto in modo che l'oggetto possa essere ripristinato in seguito a questo stato. Conosciuto anche come Token.
- Partecipanti:
  - **Memento**: memorizza lo stato interno di un oggetto.
  - **Originator**: crea un memento che rappresenta lo stato interno dell'oggetto.
  - **Caretaker**: salva e ripristina lo stato interno dell'oggetto.

- Collaborazioni: il caretaker richiede un memento all'originator e lo salva. Il caretaker può richiedere all'originator di ripristinare lo stato interno dell'oggetto.
- Conseguenze:
  - Permette di catturare e esternalizzare lo stato interno di un oggetto in modo che l'oggetto possa essere ripristinato in seguito a questo stato.
  - Può essere costoso in termini di prestazioni.
  - Difficile garantire che solo l'originator possa accedere allo stato interno memorizzato nel memento.
- Pattern correlati: Command, Iterator.

Observer (OB)

- Intento: Definire una dipendenza uno a molti tra oggetti, in modo che quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente. Anche noto come Publish-Subscribe o Dependents.
- Partecipanti:
  - **Subject**: conosce i suoi osservatori. Fornisce un'interfaccia per l'aggiunta e la rimozione di osservatori.
  - **Observer**: definisce un'interfaccia per la ricezione delle notifiche.
  - **ConcreteSubject**: invia notifiche agli osservatori quando cambia il suo stato.
  - **ConcreteObserver**: implementa l'interfaccia per la ricezione delle notifiche.
- Collaborazioni: gli osservatori registrati vengono notificati quando cambia lo stato del soggetto. Il soggetto può inviare notifiche agli osservatori. Gli osservatori possono registrarsi e cancellarsi dal soggetto.
- Conseguenze:
  - Dipendenza uno a molti tra oggetti.
  - Supporta l'invio di notifiche agli osservatori.
  - Supporta la modifica dinamica del numero di osservatori.
  - Gli osservatori possono essere notificati in ordine casuale.
  - Gli osservatori possono essere notificati anche quando non è necessario.
- Pattern correlati: Mediator, Singleton.

State (ST)

- Intento: Permette a un oggetto di modificare il suo comportamento quando cambia il suo stato interno. L'oggetto sembra cambiare la sua classe.
- Partecipanti:
  - **Context**: definisce l'interfaccia per l'interazione con lo stato.
  - **State**: definisce un'interfaccia per l'incapsulamento degli stati specifici.
  - **ConcreteState**: implementa un comportamento associato a uno stato.
- Collaborazioni: il context delega le richieste allo stato corrente. Quando lo stato cambia, il context delega le richieste allo stato nuovo.
- Conseguenze:

    - Incapsula lo stato in oggetti separati.
    - Facilita l'aggiunta di nuovi stati.
    - Rende esplicito il comportamento dipendente dallo stato.
    - Gli oggetti condivisi possono essere difficili da gestire.
    - Condivisione di oggetti di stato.
- Pattern correlati: Strategy, Flyweight.

Strategy (STG)

- Intento: Definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Consente agli algoritmi di variare indipendentemente dai client che li utilizzano.
- Partecipanti:
    - **Strategy**: definisce un'interfaccia comune per tutti gli algoritmi supportati.
    - **ConcreteStrategy**: implementa un algoritmo.
    - **Context**: utilizza un oggetto Strategy per implementare un algoritmo.
- Collaborazioni: il context invoca l'operazione definita dall'interfaccia Strategy. Il context non conosce la classe concreta che implementa l'interfaccia Strategy.
- Conseguenze:
    - Incapsula gli algoritmi in oggetti separati.
    - Facilita l'aggiunta di nuovi algoritmi.
    - Rende esplicito il comportamento dipendente dall'algoritmo.
    - Client deve conoscere le strategie.
- Pattern correlati: Flyweight.

Command (CMD)

- Intento: Incapsulare una richiesta in un oggetto, consentendo così di parametrizzare i client con diverse richieste, accodare o registrare richieste e supportare operazioni annullabili.
- Partecipanti:
    - **Command**: definisce un'interfaccia per l'esecuzione di un'operazione.
    - **ConcreteCommand**: implementa un'operazione.
    - **Client**: crea un oggetto Command e lo passa al receiver.
    - **Receiver**: sa come eseguire le operazioni associate a un comando.
    - **Invoker**: richiede l'esecuzione di un comando.
- Collaborazioni: il client crea un oggetto Command e lo passa al receiver. L'invoker richiede l'esecuzione di un comando. L'invoker non conosce la classe concreta che implementa il comando.
- Conseguenze:
    - Disaccoppiamento tra il sender di una richiesta e il receiver.
    - Supporta l'aggiunta di nuovi comandi.
    - Supporta l'annullamento delle operazioni.
- Pattern correlati: Composite, Memento.

Iterator (ITR)

- Intento: Fornire un modo per accedere agli elementi di un oggetto aggregato in sequenza senza esporre la sua rappresentazione interna.
- Partecipanti:
  - **Iterator**: definisce un'interfaccia per l'accesso e la navigazione degli elementi.
  - **ConcreteIterator**: implementa un'interfaccia per l'accesso e la navigazione degli elementi.
  - **Aggregate**: definisce un'interfaccia per la creazione di un iteratore.
  - **ConcreteAggregate**: implementa un'interfaccia per la creazione di un iteratore.
- Collaborazioni: il client utilizza l'iteratore per accedere agli elementi di un oggetto aggregato.
- Conseguenze:
  - Supporta l'accesso agli elementi di un oggetto aggregato senza esporre la sua rappresentazione interna.
  - Supporta l'accesso agli elementi di un oggetto aggregato in sequenza.
  - Supporta l'accesso agli elementi di un oggetto aggregato in sequenza in base a diverse politiche.
  - Difficile supportare più iterazioni contemporanee.
- Pattern correlati: Composite, Factory Method, Memento.

Visitor (VST)

- Intento: Definire una nuova operazione da eseguire sugli oggetti senza cambiare le classi su cui opera l'operazione.
- Partecipanti:
  - **Visitor**: definisce un'interfaccia per l'esecuzione di operazioni su un oggetto.
  - **ConcreteVisitor**: implementa un'operazione.
  - **Element**: definisce un'interfaccia per l'accettazione di un visitor.
  - **ConcreteElement**: implementa un'interfaccia per l'accettazione di un visitor.
  - **ObjectStructure**: raggruppa gli elementi e fornisce un'interfaccia per l'iterazione su di essi.
- Collaborazioni: il client crea un oggetto Visitor e lo passa all'ObjectStructure. L'ObjectStructure invoca l'operazione del Visitor su tutti gli elementi.
- Conseguenze:
  - Permette di definire nuove operazioni senza cambiare le classi su cui opera l'operazione.
  - Facilita l'aggiunta di nuove operazioni.
  - Raggruppa operazioni correlate.
  - Difficile aggiungere nuove classi Element.
- Pattern correlati: Composite, Iterator.