Esonero comportamentali

1. **Indicate i pattern fortemente correlati (1 punto)**:
    - Template, Strategy, State, Command
2. **Indicare tutti e soli i pattern comportamentali (1 punto)**:
    - Strategy, State, Visitor, Mediator, Command, Observer, Chain of Responsibility
3. **Indicare tutti e soli i pattern che consentono di definire famiglie di algoritmi intercambiabili (1 punto)**:
    - Strategy, State
4. **Indicare tutti e soli i pattern che consentono il decoupling tra sender e receiver di richieste (1 punto)**:
    - Chain of Responsibility, Mediator, Observer, State, Command
5. **Indicare tutti e soli i pattern che hanno due modalità implementative dette pull e push (1 punto)**:
    - Observer, Iterator
6. **Indicare tutti e soli i pattern che sono correlati al pattern Interpreter (1 punto)**:
    - Visitor, Composite
7. **Indicare tutti e soli i pattern che sono comportamentali e di classe (1 punto)**:
    - Template Method, Interpreter
8. **Quando il pattern Visitor è adatto? Ci può essere più di una risposta (2 punti)**:
    - Quando si ha una struttura che contiene oggetti di svariate classi e si vogliono eseguire delle operazioni, anche diverse, che dipendono dalle classi concrete di tali oggetti.
    - Quando le classi della struttura visitata non cambiano spesso.
9. **Indicare i pattern più adatti a modellare il seguente scenario (4 punti):**

    “Si vuole modellare un sistema di gestione degli elettrodomestici/componenti di una casa, un algoritmo complesso deve modificare l’apertura delle tende o dei termosifoni o delle luci rispetto alle misurazioni di veri sensori di temperature,

luminosità, posizione presenza di ospiti. Il sistema deve configurarsi facilmente prevedendo l'aggiunta e la rimozione di componenti standard. I componenti posso essere raggruppati per funzionalità. Lo stato della casa deve essere poi consultabile ed i vari elettrodomestici/componenti controllati in remoto da una app tramite una API che semplifica la programmazione delle varie istanziazioni del sistema in case differenti. (4 punti)"

- Facade: Per semplificare l'interazione con il sistema di gestione degli elettrodomestici e nascondere la complessità sottostante.
- Mediator: Per gestire le interazioni tra i vari componenti della casa, consentendo una comunicazione centralizzata.
- Composite: Per rappresentare la gerarchia di componenti all'interno della casa in modo modulare.
- Adapter: Per adattare i vari componenti standard al sistema di gestione degli elettrodomestici.
- Command: Per incapsulare le richieste dei dispositivi e consentire operazioni come l'annullamento.

Motivazione delle risposte

Domanda 1 (Pattern correlati):

- *Template, Strategy, State, Command*: Questi pattern sono fortemente correlati perché tutti e quattro trattano la separazione dell'algoritmo di un'operazione dalla sua implementazione. Il Template Method definisce uno scheletro di algoritmo, mentre Strategy, State e Command consentono di definire diverse implementazioni dell'operazione e scambiarle in fase di runtime.

Domanda 2 (Pattern comportamentali):

- *Strategy, State, Visitor, Mediator, Command, Observer, Chain of Responsibility*: Questi sono tutti pattern comportamentali perché si concentrano sull'organizzazione dei comportamenti tra gli oggetti e sulla gestione delle interazioni tra di essi.
- Gli altri pattern comportamentali sono: *Template Method*, *Interpreter*, *Iterator*, *Memento*.

Domanda 3 (Famiglie di algoritmi intercambiabili):

- *Strategy, State*: Questi pattern consentono di definire famiglie di algoritmi intercambiabili. Strategy consente di scambiare algoritmi in fase di runtime, mentre State consente di cambiare il comportamento di un oggetto a seconda del suo stato interno.

Domanda 4 (Decoupling tra sender e receiver):

- *Chain of Responsibility, Mediator, Observer, State, Command*: Questi pattern consentono il decoupling tra mittente (sender) e ricevente (receiver) di richieste.

Chain of Responsibility permette a più oggetti di gestire richieste senza che il mittente conosca il ricevente. Mediator fa da intermediario tra oggetti, evitando comunicazioni dirette. Observer permette a oggetti di registrarsi come osservatori senza che il soggetto conosca gli osservatori specifici. State cambia il comportamento dell'oggetto senza richiedere al mittente di conoscere lo stato specifico. Command incapsula richieste come oggetti indipendenti dal mittente.

Domanda 5 (Modalità pull e push):

Il pattern Observer è noto per avere due modalità implementative: "push" e "pull".

- Modalità "push": In questa modalità, il soggetto (subject) invia automaticamente gli aggiornamenti (eventi) a tutti gli osservatori (observers) quando avviene un cambiamento. Gli osservatori ricevono attivamente queste notifiche senza richiedere esplicitamente i dati dal soggetto. Questa modalità è utilizzata quando si desidera che gli osservatori siano informati in modo proattivo dei cambiamenti.

- Modalità "pull": In questa modalità, gli osservatori richiedono esplicitamente i dati dal soggetto quando desiderano aggiornamenti. Il soggetto fornisce un meccanismo per consentire agli osservatori di recuperare i dati quando ne hanno bisogno. Questa modalità è utilizzata quando si desidera che gli osservatori abbiano il controllo su quando ricevere i dati dal soggetto.

Domanda 6 (Pattern correlati al pattern Interpreter):

- *Visitor, Composite*: Questi pattern sono correlati al pattern Interpreter perché possono essere utilizzati insieme per la visita di nodi in una struttura complessa. Visitor è utilizzato per eseguire operazioni su oggetti di classi diverse, mentre Composite è spesso utilizzato per rappresentare una struttura ad albero di oggetti.

Domanda 7 (Pattern comportamentali di classe):

- *Template Method, Interpreter*: Questi sono pattern comportamentali che si concentrano sulla gestione dei comportamenti tra classi e oggetti.

Domanda 8 (Quando è adatto il pattern Visitor):

- Il pattern Visitor è adatto quando si ha una struttura che contiene oggetti di diverse classi e si vogliono eseguire operazioni, anche diverse, che dipendono dalle classi concrete di tali oggetti.
- È adatto anche quando le classi della struttura visitata non cambiano spesso, poiché aggiungere nuove classi richiederebbe la modifica del visitor.

Domanda 9 (Pattern adatti per il sistema di gestione degli elettrodomestici):

- *Facade*: Il Facade è utile per semplificare l'interazione con un sistema complesso, nascondendo la complessità sottostante. Nel caso di un sistema di gestione degli elettrodomestici, potrebbe semplificare l'interfaccia utente.

- *Mediator*: Il Mediatore è adatto quando è necessario gestire le interazioni tra molti componenti in un sistema complesso, come nel caso di elettrodomestici e sensori. Evita che i componenti comunichino direttamente.
- *Composite*: Il Composite è utile per rappresentare gerarchie di componenti, ad esempio gruppi di elettrodomestici. È utile per gestire gruppi di oggetti in modo omogeneo.
- *Adapter*: L'Adapter può essere utile per adattare i vari componenti standard al sistema di gestione degli elettrodomestici, poiché potrebbero utilizzare interfacce diverse.
- *Command*: Il Command potrebbe essere utilizzato per incapsulare le richieste dei dispositivi e consentire operazioni come l'annullamento delle azioni.

Pattern Comportamentali

1. **Chain of Responsibility**
    - Questo pattern permette di passare la richiesta lungo una catena di gestori. Ogni gestore decide se processare la richiesta o passarla al successivo nella catena, riducendo l'accoppiamento diretto tra mittente e destinatario delle richieste.
2. **Command**
    - Incapsula una richiesta come un oggetto, consentendo di parametrizzare i client con diverse richieste, code di richieste, e operazioni di annullamento. È particolarmente utile per implementare funzionalità come l'undo/redo.
3. **Interpreter**
    - Definisce un modo per interpretare la grammatica di un linguaggio, fornendo un interprete che può valutare frasi in quel linguaggio. È utile per linguaggi specializzati o per processare espressioni linguistiche in un determinato contesto.
4. **Iterator**
    - Fornisce un modo per accedere agli elementi di un oggetto aggregato (come una collezione) senza esporne la rappresentazione interna. Facilita la navigazione attraverso collezioni di vari tipi.
5. **Mediator**
    - Definisce un oggetto che centralizza la comunicazione tra oggetti collegati, promuovendo un basso accoppiamento evitando che gli oggetti si riferiscano esplicitamente l'uno all'altro.
6. **Memento**

    - Consente di salvare e ripristinare lo stato precedente di un oggetto senza rivelarne i dettagli implementativi. È comunemente usato per implementare la funzionalità di annullamento in applicazioni software.

7. **Observer**
    - Crea una relazione uno-a-molti tra oggetti in modo tale che quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente. È ampiamente usato in sistemi distribuiti per implementare la comunicazione evento-driven.

8. **State**
    - Permette a un oggetto di modificare il suo comportamento quando il suo stato interno cambia. Questo è realizzato creando oggetti di stato che rappresentano vari stati e un contesto che mantiene lo stato corrente.

9. **Strategy**
    - Definisce una famiglia di algoritmi, incapsula ciascuno di essi, e li rende intercambiabili. Strategy consente di modificare l'algoritmo utilizzato indipendentemente dai client che ne fanno uso.

10. **Template Method**
    - Definisce lo scheletro di un algoritmo all'interno di un metodo, ritardando alcuni passaggi all'implementazione da parte delle sottoclassi. Permette alle sottoclassi di modificare certi passaggi dell'algoritmo senza cambiare la sua struttura.

11. **Visitor**
    - Consente di aggiungere nuove operazioni a classi di oggetti senza modificarle. I visitatori dichiarano operazioni per ogni classe di elemento che possono visitare. Questo permette di eseguire operazioni su una collezione di oggetti eterogenei.

Correlati

1. **Finalità Comune**: I pattern sono correlati se sono progettati per risolvere problemi simili o soddisfare requisiti simili. Ad esempio, diversi pattern comportamentali possono essere correlati perché tutti mirano a gestire la comunicazione e l'interazione tra gli oggetti.

2. **Principi di Design Simili**: Se due o più pattern sono basati su principi di design simili, possono essere considerati correlati. Ad esempio, pattern che enfatizzano la separazione delle preoccupazioni o l'incapsulamento possono rientrare in questa categoria.

3. **Complementarietà**: I pattern che spesso vengono usati insieme in modo complementare sono considerati correlati. Per esempio, l'utilizzo combinato di un pattern strutturale e un pattern creazionale può offrire una soluzione completa per un particolare problema di design.

4. **Modelli di Uso Comuni**: Se nell'industria o nella comunità di sviluppo software ci sono tendenze comuni nell'utilizzo combinato di certi pattern, questi possono essere considerati correlati. Questo può essere dovuto alla loro efficacia combinata nel risolvere specifici problemi di design.

5. **Variazioni sullo Stesso Tema**: Alcuni pattern possono essere considerati variazioni dello stesso tema di base. Ad esempio, diversi pattern creazionali affrontano il problema della creazione di oggetti, ma con approcci leggermente diversi.

6. **Interdipendenza Funzionale**: Se l'implementazione o l'efficacia di un pattern dipende o è migliorata dall'uso di un altro pattern, questi possono essere considerati correlati. Ad esempio, l'implementazione di un pattern può prevedere l'utilizzo intrinseco di un altro pattern.

7. **Storia e Evoluzione**: La storia e l'evoluzione dei pattern possono anche indicare una correlazione. Ad esempio, se un pattern è stato sviluppato come una risposta o una raffinazione di un altro, o se entrambi sono emersi da un concetto di design comune, possono essere considerati correlati.

Fortemente correlati

1. **Soluzione a Problemi Simili**: I pattern sono considerati fortemente correlati se forniscono soluzioni a problemi molto simili o se sono comunemente usati per affrontare lo stesso tipo di sfida nel design del software.

2. **Complementarietà**: Due o più pattern sono fortemente correlati se si completano a vicenda in modo tale che l'uso combinato dei pattern fornisca una soluzione più robusta o efficace rispetto all'uso di un singolo pattern.

3. **Principi di Design Condivisi**: I pattern possono essere fortemente correlati se si basano su principi di design simili. Ad esempio, pattern che promuovono l'alta coesione e il basso accoppiamento potrebbero essere visti come correlati in questo senso.

4. **Relazioni Strutturali o Funzionali**: Se i pattern hanno strutture o meccanismi interni simili, o se uno pattern necessariamente implica o facilita l'uso dell'altro, possono essere considerati fortemente correlati.

5. **Frequenza di Uso Combinato**: Alcuni pattern sono frequentemente usati insieme nella pratica. Questa tendenza all'uso combinato può indicare una forte correlazione, poiché i designer di software trovano che questi pattern si integrano bene l'uno con l'altro.

6. **Evoluzione dei Pattern**: In alcuni casi, i pattern possono essere considerati fortemente correlati se uno è una variante o un'estensione dell'altro, o se entrambi hanno origini comuni in termini di concetti di design.

Correlati dei pattern comportamentali

1. **Chain of Responsibility**
    - **Correlati**: Command, Mediator
    - **Perché**: È simile a Command in quanto entrambi decouplano il sender dal receiver. Si relaziona anche a Mediator in quanto può essere visto come una catena di mediatori.
2. **Command**
    - **Correlati**: Memento, Strategy, Chain of Responsibility
    - **Perché**: Si collega a Memento poiché può utilizzare questo pattern per implementare l'undo/redo. Strategy può essere usato per variare l'implementazione dei comandi. Chain of Responsibility può gestire i comandi in una sequenza.
3. **Interpreter**
    - **Correlati**: Composite, Visitor
    - **Perché**: Spesso utilizza Composite per rappresentare la struttura grammaticale. Visitor può essere usato per eseguire operazioni sull'interprete dell'albero astratto.
4. **Iterator**
    - **Correlati**: Memento, Composite
    - **Perché**: Memento può essere utilizzato per salvare lo stato dell'iterazione. Composite, perché l'Iterator è spesso usato per navigare attraverso strutture Composite.
5. **Mediator**
    - **Correlati**: Observer, Chain of Responsibility
    - **Perché**: Observer è correlato in quanto entrambi facilitano la comunicazione tra oggetti. Chain of Responsibility può essere considerata una catena di mediatori.
6. **Memento**
    - **Correlati**: Command, Iterator
    - **Perché**: Si collega a Command per supportare l'undo/redo dei comandi. È correlato a Iterator per salvare lo stato dell'iterazione.
7. **Observer**

- **Correlati**: Mediator, State, Strategy
- **Perché**: Mediator e Observer entrambi trattano la comunicazione tra oggetti. State e Strategy sono correlati in quanto possono essere utilizzati per cambiare il comportamento dell'oggetto osservato.

8. **State**

   - **Correlati**: Strategy, Observer
   - **Perché**: È simile a Strategy poiché entrambi modificano il comportamento degli oggetti. Observer può essere utilizzato per notificare i cambiamenti di stato.

9. **Strategy**

   - **Correlati**: State, Command
   - **Perché**: Simile a State in quanto entrambi modificano il comportamento degli oggetti. Si collega a Command poiché le strategie possono essere utilizzate per definire vari comportamenti di un comando.

10. **Template Method**

    - **Correlati**: Strategy, Factory Method
    - **Perché**: Strategy può offrire un approccio più flessibile per alcune delle fasi in Template Method. Factory Method può essere utilizzato all'interno di un Template Method per delegare la creazione di oggetti.

11. **Visitor**

    - **Correlati**: Interpreter, Composite
    - **Perché**: È spesso usato con Composite per applicare operazioni agli elementi di una struttura composita. Si collega a Interpreter poiché può essere utilizzato per eseguire operazioni su un albero di interpretazione.