UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

6. Design Pattern in Java

1 esercizio risolto + 2 extra

Tutorato di
Programmazione Avanzata

RELATORE
Imberti Federico

SEDE
Dalmine, BG

Key-points della teoria: Design Pattern

1. Permettono di ripetere soluzioni efficaci quando si incontrano problemi simili tra loro: inutile reinventare la ruota!
2. Totale di 23 pattern divisi In base

All'**obiettivo** in 3 famiglie:

- Creazionali
- Strutturali
- Comportamentali

Allo **scope** in 2 famiglie:

- Classi
- Oggetti

Pattern Creazionali: Singleton

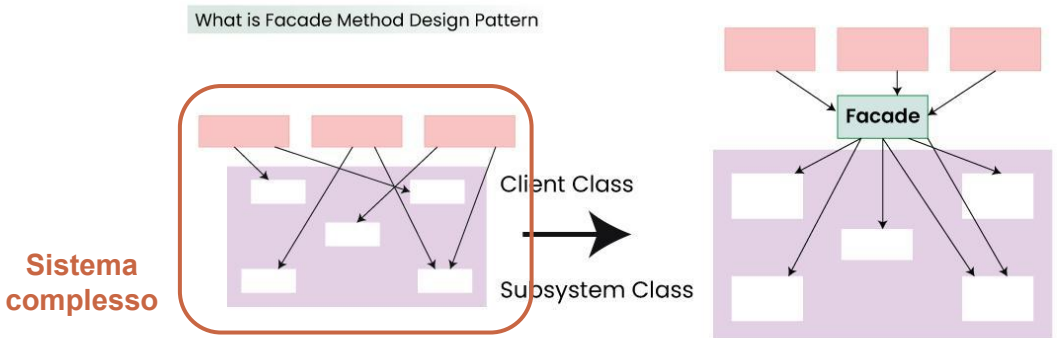
- Permette di limitare il numero di istanza possibili di una classe. Generalmente il limite viene posto a 1;

- Possibile implementarlo in versione “eager” o “lazy”
  - Cambia il momento in cui l’istanza viene generata: nel primo caso al lancio del programma mentre nel secondo solo quando serve l’istanza per la prima volta

Singleton
- Singleton() + static getInstance() -> Singleton
- static instance: Singleton

Pattern Strutturali: Facade

- Permette di astrarre la struttura di un sistema complesso attraverso un oggetto intermedio con cui si interfaccia l'utente finale
  - Semplificare interfaccia del sistema complesso…
  - Riduciamo coupling;
  - Aumentiamo manutenibilità;



Pattern Comportamentali: Visitor

- Permette di aggiungere temporaneamente un metodo a una classe “visitabile” mediante dei “visitatori”

    - La classe visitabile deve implementare l’interfaccia “**Visitable**” che dispone di definire un metodo pubblico “**accept(visitor: Visitor)**” per accettare visitatori. L’unico compito di “accept” sarà di eseguire il metodo “visiti()” del visitatore passato in input (accettato).

    - I visitatori dovranno implementare l’interfaccia “**Visitor**” che dispone di implementare un metodo pubblico “**visit()**”. Si tratta del metodo “aggiunto temporaneamente” alla classe!

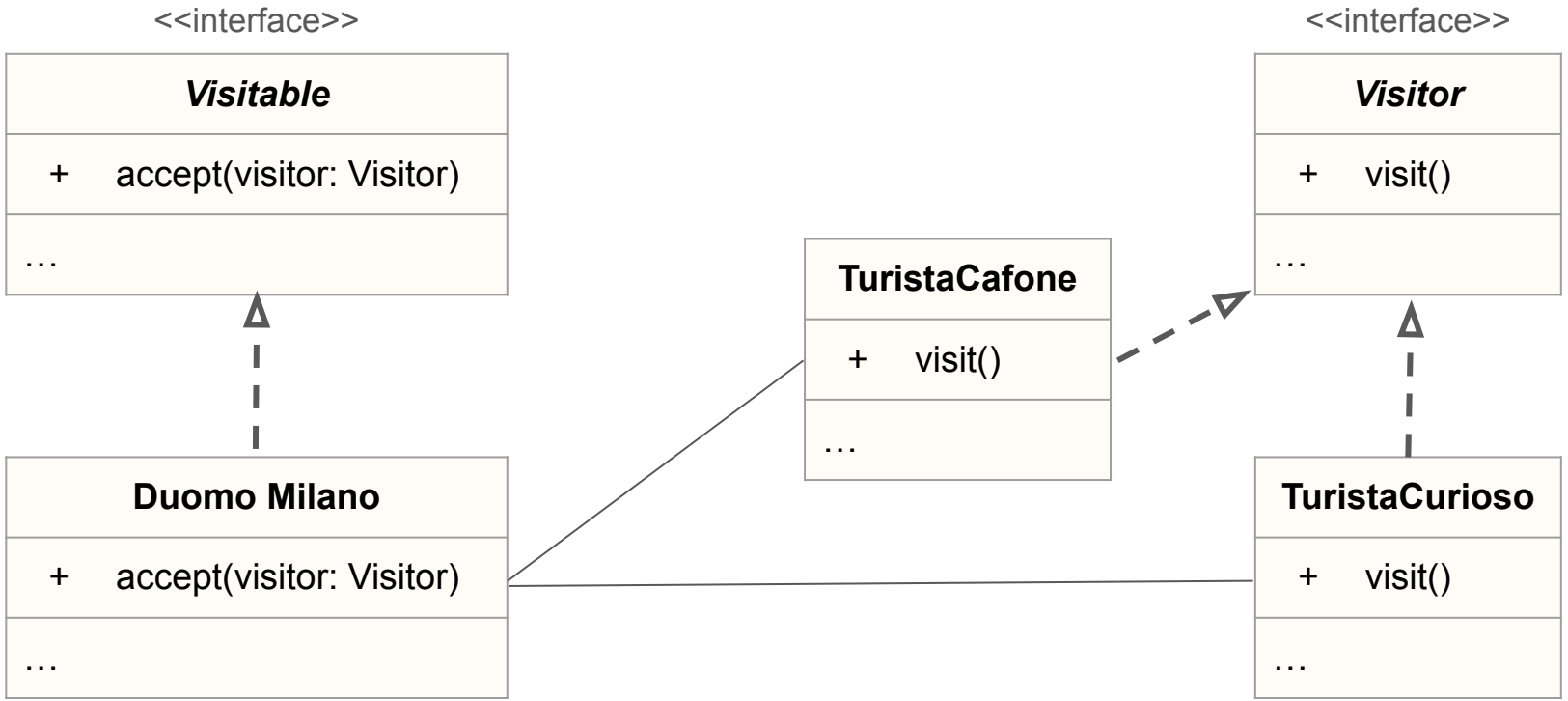
Pattern Comportamentali: Esempio di Visitor

*Modelliamo tipologie di **turisti** vogliono che vogliono visitare **monumenti** famosi.*

Per accettare turisti dobbiamo fare in modo che :

- I monumenti siano visitabili e in grado di accettare visitatori
  - implementano Visitable con il normale metodo “visit()”
- I visitatori abbiano la capacità di visitare un monumento come meglio preferiscono
  - Implementano Visitor con un metodo “visit()” personalizzato per ogni tipologia di turista

Pattern Comportamentali: Esempio di Visitor



UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Domandine?

Esercizio 1/1

Un Animale può essere un Pesce. Per vedere se un animale nuota oppure no, invece di usare un metodo, voglio usare il visitor pattern. Assumo che ogni pesce nuoti e che ogni animale che nuota è un pesce – cioè un animale che non è un pesce non nuota. Come faresti?
Se volessi scrivere un metodo statico che data una lista di animali, conta quanti nuotano, come lo implementeresti?
Se riesci, evita l'uso di instaceof

Esercizio Extra 1/2

Si vuole gestire il menu di un ristorante. Il ristorante offre varie pietanze. Ogni pietanza ha un quantitativo di calorie. Considerare due pietanze di esempio: le carote (carrot) e l'agnello (lamb). A differenza delle carote, l'agnello ha un peso ed è possibile selezionare la cottura mediante un opportuno metodo: l'agnello offre il metodo configuraCottura(String cotturaSelezionata). Le carote offrono un metodo configuraPrezzemolo che prende un booleano e permette di inserire o meno il prezzemolo nelle carote.

(a) Si vuole utilizzare il pattern Visitor per
    1) stampare il menu in varie lingue (considerare per esempio l'inglese e l'italiano) e
    2) stampare le calorie dei vari piatti

(b) Deve essere possibile ordinare le pietanze in ordine crescente in base alle loro calorie. Utilizzare l'interfaccia Comparable e mostrare un esempio nel quale viene creata una lista di pietanze e vengono ordinate in base alle calorie.

Esercizio Extra 2/2

Si vuole sviluppare un'applicazione per gestire una linea ferroviaria. Ci sono tre tipologie di treni: alta velocità, interregionali, locali. Ogni treno ha una partenza e una destinazione. A differenza dei treni ad alta velocità che si fermano solo alla partenza e alla destinazione, i treni interregionali e locali hanno una serie di fermate intermedie. I treni ad alta velocità, interregionali e locali hanno un massimo di 100, 150 e 200 passeggeri. Una corsa è effettuata da un treno e ha un numero totale di passeggeri che hanno preso il treno durante quella corsa (nota il numero potrebbe essere più alto nel caso di treni interregionali e locali considerato che i passeggeri possono salire e scendere nelle fermate intermedie).

Si vuole utilizzare il pattern Visitor per (a) stampare le fermate dei vari treni: per i treni ad alta velocità il nome delle fermate deve essere preceduta dalla stringa "AV -", per i treni interregionali il nome delle fermate deve essere preceduto dalla stringa "I -", per i treni locali il nome delle fermate deve essere preceduto dalla stringa "L -". (b) stampare il numero massimo dei passeggeri dei vari treni.

Deve essere possibile ordinare le corse in ordine crescente in base al numero di passeggeri. Utilizzare l'interfaccia Comparable e mostrare un esempio nel quale viene creata una lista di corse e vengono ordinate in base ai passeggeri.

Creare una Lista di elementi di treni interregionali. Discuti se è possibile assegnare questa lista ad un Lista di elementi di tipo Treno e discuti le relative motivazioni.