28-03-2023

Hash Map

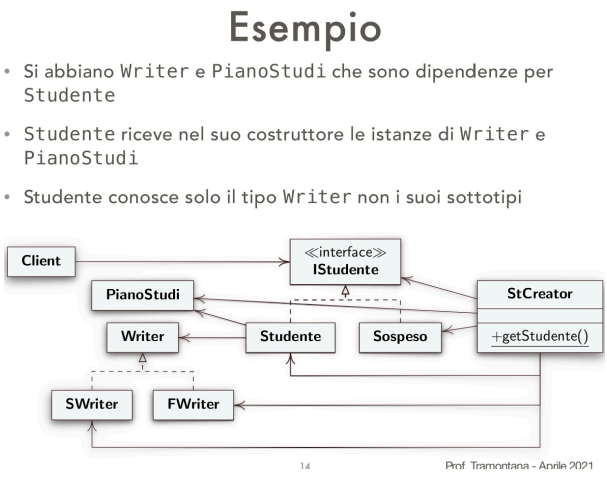
`HasMap` è una mappa associativa, formata da coppie di valori: **il primo valore** è la **chiave di accesso** per il secondo valore
Operazioni

- Inserimento in tabella: `put(chiave,valore)`
- Trovare il valore in base a una chiave: `valore= tabella.get(chiave)`
- Rimuovere un valore dalla tabella: `valore=tabella.remove(chiave)`

Aver separato Creator e ConcreteCreator introduce il meccanismo Dependency Injection: La classe MainEsami crea istanza di StCreator e la fa conoscere al Client. Il Client si lega e Creator e non a StCreator. In particolare:

```
public class MainEsami{
        public static void main(String[] args){
                Creator crea = new StCreator();
                Clienc c = new Client(crea);
        }
}
```

- Creo una sola istanza di StCreator e la passo al Client
- Si separa la decisone sulla costruzione dall'uso di quell'istanza
- Il client non deve sapere le istanze create ma deve conoscere solo l'interfaccia. (incapsulamento)
- Le dipendenze sono iniettate al client per mezzo di parametri nel suo costruttore. Questo permette di evitare complicazioni derivanti da metodi setter e da controlli per verificare che le dipendenze non siano null, di conseguenza il codice è più semplice
- L'oggetto che fa Dependency Injection si occupa di connettere (*fa **wiring** di*) varie istanze. In un unico posto vediamo le connessioni fra gli oggetti.



Abstract Factory

Intento

Fornire un'interfaccia per creare famiglie di oggetti che hanno qualche relazione senza specificare le loro classi concrete

- Si hanno più gerarchie di Product e ConcreteProduct.
- Product è un'interfaccia totalmente diversa da factory method
- Quind ho **più interfacce** di Product:
  - un'interfaccia **ProductB** che implementa **ConcreteProductB1** e **ConcreteProductB2**
  - un'interfaccia **ProductA** che implemente **ConcreteProductA1** e **ConcreteProductA2**
- Quando seleziono ConcreteProductA2 è giusto **avere una corrispondenza** con ConcreteProductB2 (o anche altre relazioni)
  - Quando uso B2 mi relaziono solo a A2 o viceversa e non relazionarsi a A1 o B1

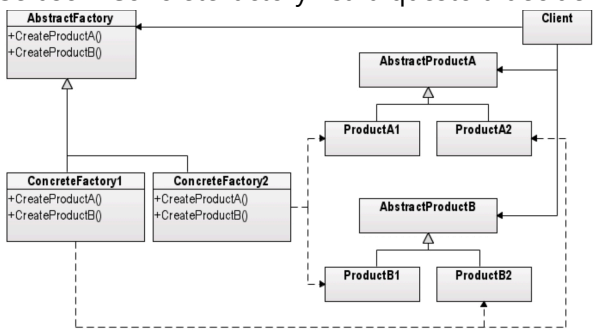
Problema

- Il sistema complessivo dovrebbe essere indipendente dalle classi usae, così da essere configurabile con una di varie famiglie di classi. Le classi di una famiglia dovrebbero essere usate insieme (in modo consistente)

> *Es. lo strato di interfaccia utente permette vari tipi di look-and-feel*

Soluzione

1. Creare delle gerarchie con interfacce AbstracProductB e AbstractProductA
2. Gestire la parte di creazione di istanze AbstractFactory e ConcreteFactory
3. Se uso il ConcreteFactory1 sarà questo a decidere la consistenza.



Usare una **serie di ruoli**: AbstractFactory, ConcreteFactory, AbstractProduct

- Si possono creare istanze di famiglie di classi in maniera consistente.
- Le famiglie di classi sono facilmente intercambiabili.
  - Se voglio eliminare una famiglia di AbstractProduct e cambiare (Per esempio un bottone cliccabile) in un bottone che scompare, allora posso inserire una nuova famiglia con un'interfaccia. In questo caso devo solo modificare il ConcreteFactory far istaziare le nuove classi create.

Maggiori informazioni su Wiki

> *Se volessi, quindi, AbstractProductC basta fare le operazioni sopra citate, quindi creare una nuova gerarchia.*

- **Uso di classi di famiglie diverse corrisponde a diverse famiglie di classi.**

Object Pool

Si ha bisogno di tenere le istanze (in una lista) create a run-time per riusare la stessa istanza in circostanze diverse. Questa tecnica è detta **Objcet Pool**

> *Quando si finisce di usare un'istanza, potrebbe venir restituita (venendo **liberata**) e **riutilizzata**.*

Utile per:

- Creazione di istanze può essere **pesante a livello computazionale**.
- Quando le istanze sono "libere" allora possono essere fornite a chi vuole una nuova istanza
- Si evita di creare altri spazi di memoria

Serve:

- Un posto per tenere le istanze (lista delle istanze create) messa in una classe che ha appunto tale ruolo (all'interno del factoryMethod).

```java
import java.util.LinkedList;
// CreatorPool è un ConcreteCreator e implementa un Object Pool
public class CreatorPool extends ShapeCreator {
  private LinkedList<Shape> pool = new LinkedList<Shape>();
  // getShape() è un metodo factory che ritorna un oggetto preleavto dal pool
  public Shape getShape() {
      Shape s;
      if (pool.size() > 0) s = pool.remove();
      else s = new Circle();
      return s;
  }
  // releaseShape() inserisce un oggetto nel pool
  public void releaseShape(Shape s) {
      pool.add(s);
  }
}
```

E. Tramontana Pool, Dependency - 12 May 10 2

- Queste tecnica può **non avere limiti** oppure **può averne** (in base ai requisiti)
-