Di Dario Frongillo

19 Dicembre 2017

2 Comments

# INTRODUZIONE

Continuando il discorso sui pattern creazionali iniziato con il builder pattern, oggi vi parlaremo del **Factory Method** pattern. Il **GoF** definisce il factory method pattern nel seguente modo:

Di Dario Frongillo
19 Dicembre 2017

2 Comments

Definisce un interfaccia per creare oggetti, ma lascia alle sottoclassi la decisione del tipo di classe a istanziare.

Il pattern può rivelarsi utile quando una classe non è in grado di conoscere a priori il tipo di oggetti da creare o  quando si vuole delegare la creazione di un oggetto alle sottoclassi. L'applicazione del pattern consente di eliminare le dipendenze dai tipi concreti utilizzati.

## COMPONENTI E STRUTTURA

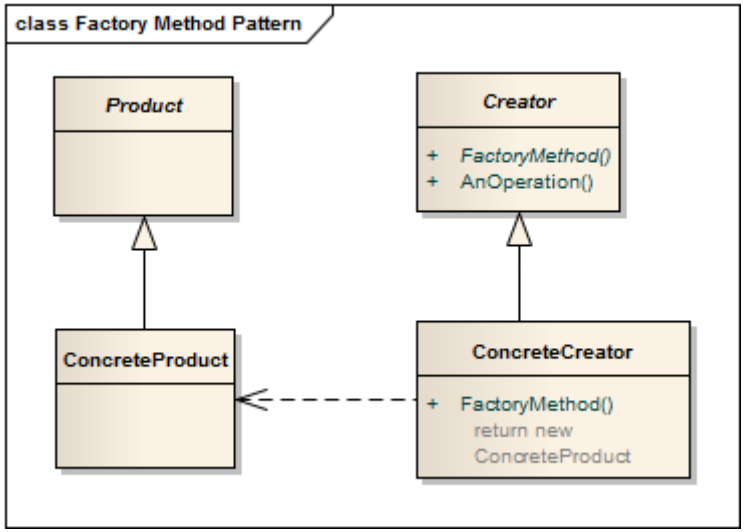
In questo  pattern possiamo individuare i seguenti componenti:

Di Dario Frongillo
19 Dicembre 2017

al fine di ritornare l'implementazione dell'oggetto

3. **Product**: definisce l'interfaccia dell'oggetto che deve essere creato dalla Factory
4. **ConcreteProduct**: implementa l'oggetto in base ai metodi definiti dall'interfaccia Product.

Se consultiamo il GoF o altra documentazione on line, troviamo la seguente rappresentazione architetturale del pattern:




# IMPLEMENTAZIONE

Di Dario Frongillo
19 Dicembre 2017

La Super Class nel factory method design pattern puo' essere un interfaccia, classe astratta o una normale classe. Nel nostro caso prendiamo ad esempio l'interfaccia animal

```
1.  public interface  Animal {
2.      String getCall();
3.  }
```

Le classi implementeranno Animal dovranno fornire un implementazione del metodo getCall: metodo per ottenere il verso di un animale il quale sara' differente per ogni specie.

```
1.  public class Dog implements Animal {
2.      @Override
3.      public String getCall() {
4.          return "Bau";
5.      }
6.  }
```

```
1.  public class Cat implements Animal {
2.      @Override
3.      public String getCall() {
4.          return "Miao";
5.      }
6.  }
```

Di seguito abbiamo l'implementazioni della Factory: la quale crea la specie giusta di animale in base all'enumerato specificato in input.

```
1.  public class AnimalFactory {
2.      public AnimalFactory(){}
3.
4.      public Animal getAnimal (AnimalEnum type){
5.          Animal retval = null;
6.          switch (type){
7.              case Cat:
8.                  retval = new Cat();
9.                  break;
10.             case Dog:
11.                 retval = new Dog();
12.                 break;
13.         }
14.         return retval;
15.     }
16. }
```

Riportiamo un semplice main che per ogni valore dell'enumerato crea una specie dell'animale associato stampando a video il verso.

```
1.  import java.util.Arrays;
2.
3.  public class Main {
4.
5.      public static void main(String[] args) {
6.
7.          AnimalFactory factory = new AnimalFactory();
8.          Arrays.stream(
9.                  AnimalEnum.values()).forEach(
10.                 type-> System.out.println("Il verso e' "+
    factory.getAnimal(type).getCall()));
11.
12.
13.     }
14. }
```

Di Dario Frongillo
19 Dicembre 2017

# APPLICABILITA'

Riassumendo l'utilizzabilita' del pattern factory method, esso e' molto utile quando:

- Una classe non è in grado di sapere in anticipo le classi di oggetti che deve creare.
- La creazione di un oggetto preclude il suo riuso senza una significativa duplicazione di codice.
- La creazione di un oggetto richiede l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione.
- La gestione del ciclo di vita degli oggetti gestiti deve essere centralizzata in modo da assicurare un comportamento coerente all'interno dell'applicazione.
- Le classi delegano la responsabilità di creazione.

factory java OOP pattern

Di Dario Frongillo
19 Dicembre 2017

## A PROPOSITO DI ME

### DARIO FRONGILLO

Fondatore di Italiancoders e dell iniziativa devtalks. Software architect con un forte background in Java, Architettura REST, framework Spring , Database Design,progettazione e sviluppo di SPA e RIA Web application con framework Javascript. Attualmente mi occupo di sviluppo soluzioni software in ambito Banking e Finance: in particolare progettazione e sviluppo di applicativi web based per la realizzazione di sistemi di trading, interfacce con i mercati finanziari e di servizi real time.

ALTRO

Di Dario Frongillo
19 Dicembre 2017

2 Comments

Di Dario Frongillo
19 Dicembre 2017

2 Comments

1. Quale sarebbe la classe che non è in grado di sapere in anticipo le classi di oggetti che deve creare?

2. Dove evitiamo che "La creazione di un oggetto preclude il suo riuso senza una significativa duplicazione di codice."

3. Quale sarebbe la classe di composizione di "La creazione di un oggetto richiede l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione."

4. Presumo che con ciclo di vita qui non abbiamo niente a che fare

5. Quale classe delega a chi (Le classi delegano la responsabilità di creazione)

// -----

A me sembra che nel esempio abbiamo soltanto un ConcreteCreator e nella mancanza di Creator non si vedono proprio i vantaggi del Factory Method pattern, anzi, direi che NON LO ABBIAMO IMPLEMENTATO.

• Rispondi • Condividi ›

**Mattia Marchiani** • 3 anni fa

Ottimo articolo, bravo Dario!
Personalmente uso frequentemente il patter Factory Method, nelle web app sviluppate a lavoro: permette di astrarre la creazione dei DTO scegliendo che tipo di struttura istanziare a runtime. Inoltre con un pizzico di Dependecy Injection si può astrarre la classe che utilizzerà la factory dalla sua reale implementazione (con una classica interfaccia IFactory<>) permettendo una flessibilità

Di Dario Frongillo
19 Dicembre 2017

2 Comments

I NOSTRI PARTNER


Privacy Policy

Di Dario Frongillo
19 Dicembre 2017

2 Comments