



Di Alessio Fiore
10 Febbraio 2018

2 Comments



INTRODUZIONE

Dopo aver parlato di Factory Method, Singleton e Build Pattern trattiamo un altro design pattern tra quelli classificati come creazionali dalla **GoF**, ovvero **Abstract Factory** (aka Kit).

L'intento di questo pattern è il seguente:



Di Alessio Fiore
10 Febbraio 2018

2 Comments

Forinare un interfaccia per creare

famiglie di oggetti correlati o
dipendenti senza specificare le loro
classi concrete limitandone quindi
l'uso diretto.



MOTIVAZIONI

In alcuni casi può essere necessario utilizzare un sistema in diversi contesti con il conseguente utilizzo di classi differenti, ma della stessa "famiglia".

L'utilizzo diretto delle classi concrete crea un accoppiamento forte e limita la portabilità del sistema stesso.

Questo porta alla necessità di rendere il sistema indipendente dalla modalità di creazione delle classi concrete, facendo sì che soltanto le interfacce siano note e non le implementazioni.

Il pattern Abstract Factory consente di rendere tra loro interscambiabili le diverse implementazioni che soddisfano una



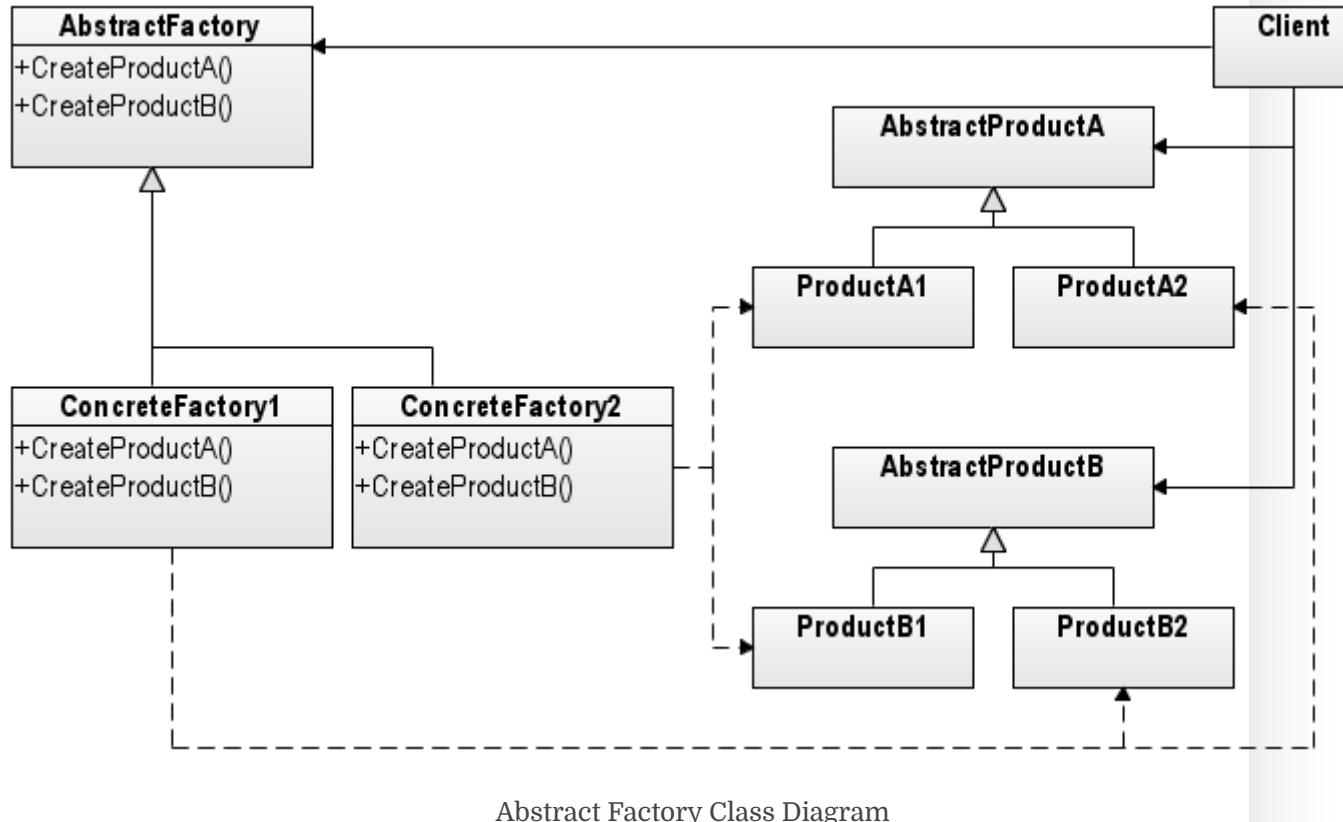
Di Alessio Fiore
10 Febbraio 2018

2 Comments

STRUTTURA



Prendiamo come riferimento della struttura il **class diagram** definito dalla GoF.



PARTECIPANTI



Di Alessio Fiore
10 Febbraio 2018

2 Comments

AbstractFactory (AbstractFactory)

Definisce l'interfaccia di riferimento per gli oggetti che creano le istanze.

ConcreteFactory (ShapeFactory)

Implementa in modo concreto l'interfaccia definita da AbstractFactory e crea effettivamente una tipologia specifica di oggetti appartenenti ad una famiglia.

AbstractProduct (Shape)

Definisce l'interfaccia di riferimento per una famiglia di oggetti da creare tramite il factory corrispondente.

ConcreteProduct (Circle, Square e Rectangle)

Implementa in modo concreto l'oggetto appartenente alla famiglia per cui vale l'interfaccia AbstractProduct e che viene creato dall'oggetto factory corrispondente.

Client (Program)

Utilizza unicamente le classi astratte del factory e dell'oggetto da creare, senza conoscerne gli aspetti implementativi.
L'annullamento dell'accoppiamento tra il client e gli oggetti



Di Alessio Fiore
10 Febbraio 2018

2 Comments

IMPLEMENTAZIONE



Consideriamo a titolo di esempio la famiglia di oggetti **Shape**.

Vogliamo far in modo che sia possibile istanziare oggetti di diversa “forma” senza utilizzare direttamente le rispettive classi concrete, ma utilizzando l’AbstractFactory.

Vediamo di seguito l’implementazione in Java dei vari elementi discussi sopra.

```
1. public interface Shape {  
2.     void draw();  
3. }
```

```
1. public class Rectangle implements Shape {  
2.  
3.     @Override  
4.     public void draw() {  
5.         System.out.println("Draw a Rectangle");  
6.     }  
7. }
```

```
1. public class Square implements Shape {  
2.  
3.     @Override  
4.     public void draw() {  
5.         System.out.println("Draw a Square");  
6.     }  
7. }
```

```
1. public class RoundedRectangle implements Shape {
```



Di Alessio Fiore
10 Febbraio 2018

2 Comments

```
1. public class RoundedSquare implements Shape {  
2.     @Override public void draw() {  
3.         System.out.println("Draw a RoundedSquare");  
4.     }  
5. }
```



```
1. public abstract class AbstractFactory {  
2.     abstract Shape getShape(String shape);  
3. }
```

```
1. public class ShapeFactory extends AbstractFactory {  
2.  
3.     @Override  
4.     public Shape getShape(String shapeType) {  
5.  
6.         if(shapeType == null){  
7.             return null;  
8.         }  
9.  
10.        if(shapeType.equalsIgnoreCase("RECTANGLE")){  
11.            return new Rectangle();  
12.        }  
13.        else if(shapeType.equalsIgnoreCase("SQUARE")){  
14.            return new Square();  
15.        }  
16.  
17.        return null;  
18.    }  
19. }
```

```
1. public class RoundedShapeFactory extends AbstractFactory {  
2.  
3.     @Override  
4.     public Shape getShape(String shapeType) {  
5.  
6.         if(shapeType == null){  
7.             return null;  
8.         }  
9.  
10.        if(shapeType.equalsIgnoreCase("RECTANGLE")){  
11.            return new RoundedRectangle();  
12.        }  
13.    }
```



Di Alessio Fiore
10 Febbraio 2018

2 Comments

```

1. public class FactoryProducer {
2.     public static AbstractFactory getFactory(boolean rounded) {
3.         if(rounded) {
4.             return new RoundedShapeFactory();
5.         }else{
6.             return new ShapeFactory();
7.         }
8.     }
9. }
```

```

1. public class Client {
2.     public static void main(String[] args) {
3.         AbstractFactory shapeFactory =
4. FactoryProducer.getFactory(false);
5.         Shape shape1 = shapeFactory.getShape("RECTANGLE");
6.         shape1.draw();
7.         Shape shape2 = shapeFactory.getShape("SQUARE");
8.         shape2.draw();
9.
10.        AbstractFactory shapeFactory1 =
11. FactoryProducer.getFactory(true);
12.        Shape shape3 = shapeFactory1.getShape("RECTANGLE");
13.        shape3.draw();
14.        Shape shape4 = shapeFactory1.getShape("SQUARE");
15.        shape4.draw();
16.    }
17. }
```

L'output del codice sopra sarà il seguente:

```

1. Draw a Rectangle
2. Draw a Square
3. Draw a RoundedRectangle
4. Draw a RoundedSquare
```

CONSEGUENZE



Di Alessio Fiore
10 Febbraio 2018

2 Comments

- **Isolamento delle classi concrete.** Aiuta a controllare le classi di oggetti creati da un'applicazione. Poiché un factory incapsula la responsabilità e il processo di creazione di oggetti e isola i client dalle classi concrete.
- **Facilità lo scambio di oggetti della stessa famiglia.** Una classe concreta appare una sola volta nel codice, quando viene istanziata dal factory che è l'unico oggetto ad averne il controllo. Basta quindi modificare la ConcreteFactory.
- **Promuove la consistenza tra oggetti della stessa famiglia.** Semplifica la cooperazione tra oggetti della stessa famiglia.

SVANTAGGI

- **Supportare nuovi tipi di prodotto è difficile.** Dato che AbstractFactory definisce tutte le varie tipologie di prodotti che è possibile istanziare, aggiungere una famiglia significa modificare l'interfaccia della factory. La modifica si ripercuote a cascata nelle factory concrete e in tutte le sottoclassi, rendendo laboriosa l'operazione.



Di Alessio Fiore
10 Febbraio 2018

2 Comments

A PROPOSITO DI ME



ALESSIO FIORE

Grande appassionato di sviluppo software e linguaggi di programmazione ha una grande conoscenza nel mondo Java e framework come Spring e Hibernate.

Laureato magistrale in Ingegneria Informatica lavora come sviluppatore software in ambito Telco principalmente lato backend, ma coltivando sempre molto interesse su tecnologie frontend.

Ha una grande esperienza su system integration, architetture software e sistemi distribuiti.



Di Alessio Fiore
10 Febbraio 2018

2 Comments



Di Alessio Fiore
10 Febbraio 2018

2 Comments



Ora: investire 250€ in Amazon potrebbe darti un secondo reddito!

T1Markets

In pochi secondi puoi trovare articoli specifici al prezzo più basso

PriceZoom

Le razze di cani più costose al mondo

Il Mondo dei Cani

Azioni Amazon: con soli 250€ potresti ottenere uno stipendio fisso. Scoprilo!

ROInvesting

Many failed before. Will you complete the Trial?

2 Commenti

ItalianCoders

normativa sulla privacy

1

Accedi

Consiglia

Tweet

Condividi

Ordina dal più recente



Partecipa alla discussione...

ENTRA CON

O REGISTRATI SU DISQUS



Nome



Di Alessio Fiore
10 Febbraio 2018

2 Comments

L'implementazione fornita però mi sembra esemplifichi il Factory Method.

^ | v • Rispondi • Condividi ›



Alessio Fiore → Mike • un anno fa

Si hai ragione, si trattava di un refuso. Grazie della segnalazione

^ | v • Rispondi • Condividi ›



I NOSTRI PARTNER



Di Alessio Fiore
10 Febbraio 2018

2 Comments



Privacy Policy



Di Alessio Fiore
10 Febbraio 2018

2 Comments