Factory Method Design Pattern

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:

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

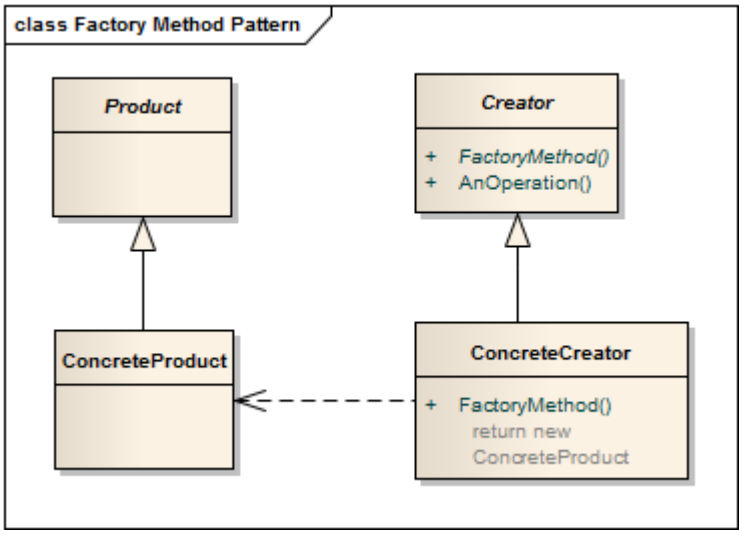
Il pattern può rivelarsi utile quando una classe non è in grado di conoscere a priori il tipo di oggetti da creare o quando si vuole delegare la creazione di un oggetto alle sottoclassi. L'applicazione del pattern consente di eliminare le dipendenze dai tipi concreti utilizzati.

COMPONENTI E STRUTTURA

In questo pattern possiamo individuare i seguenti componenti:

1. **Creator**: dichiara la Factory che avrà il compito di ritornare l'oggetto appropriato
2. **ConcreteCreator**: effettua l'overwrite del metodo della Factory 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

Per completare l'esercizio riporteremo un semplice esempio di factory method design Pattern in Java. Supponiamo di avere un dominio riguardante il mondo degli Animali.

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

```
public interface Animal { String getCall();
}
```

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.

```
public class Dog implements Animal { @Override public String
getCall() { return "Bau"; }
}
```

```
public class Cat implements Animal { @Override public String
getCall() { return "Miao"; }
}
```

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

```
public enum AnimalEnum { Cat, Dog
}

public class AnimalFactory { public AnimalFactory(){} public Animal getAnimal (AnimalEnum type){ Animal retval = null; switch (type){ case Cat: retval = new Cat(); break; case Dog: retval = new Dog(); break; } return retval; }
}
```

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

```
import java.util.Arrays; public class Main { public static void main(String[] args) { AnimalFactory factory = new AnimalFactory(); Arrays.stream( AnimalEnum.values()).forEach( type-> System.out.println("Il verso e' "+ factory.getAnimal(type).getCall())); }
}
```

```
Il verso e' Miao
Il verso e' Bau
```

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.

https://italiancoders.it/factory-method-design-pattern/