1. Abstract Factory

(GoF pag. 87)

1.1. Descrizione

Presenta un'interfaccia per la creazione di famiglie di prodotti, in modo tale che il cliente che gli utilizza non abbia conoscenza delle loro concrete classi. Questo consente:

- Assicurarsi che il cliente crei soltanto prodotti vincolati fra di loro.
- L'utilizzo di diverse famiglie di prodotti da parte dello stesso cliente.

1.2. Esempio

Si pensi a un posto di vendita di sistemi Hi-Fi, dove si eseguono dimostrazioni dell'utilizzo di famiglie complete di prodotti. Specificamente si ipotizzi che esistono due famiglie di prodotti, basate su tecnologie diverse: una famiglia che ha come supporto il nastro (*tape*), e un'altra famiglia che ha come supporto il compact disc. In entrambi casi, ogni famiglia è composta dal supporto stesso (tape o cd), un masterizzatore (*recorder*) e un riproduttore (*player*).

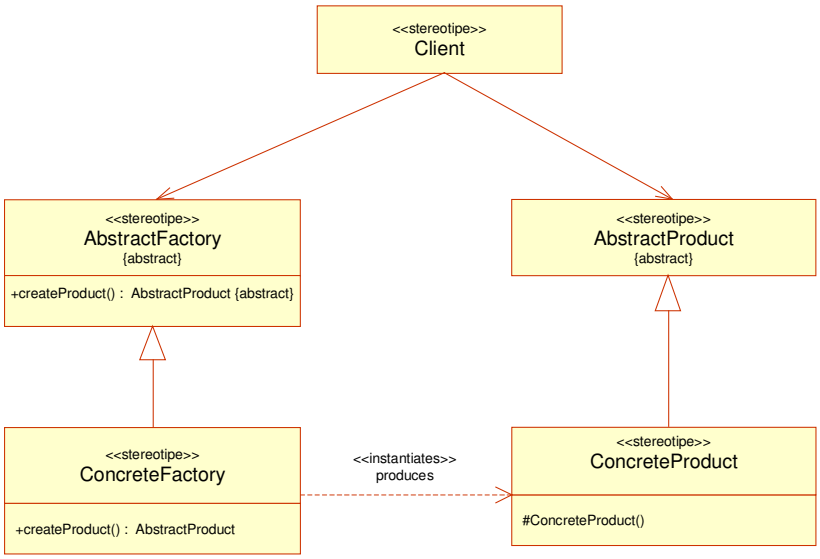
Se si accetta il fatto che questi prodotti offrono agli utenti una stessa interfaccia (cosa non tanto distante dalla realtà), un cliente potrebbe essere in grado di eseguire lo stesso processo di prova su prodotti di entrambe famiglie di prodotti. Ad esempio, potrebbe eseguire prima una registrazione nel recorder, per poi dopo ascoltarla nel player.

Il problema consiste nella definizione di un modo di creare famiglie complete di prodotti, senza vincolare alla codifica del cliente che gli utilizza, il codice delle particolari famiglie.

1.3. Descrizione della soluzione offerta dal pattern

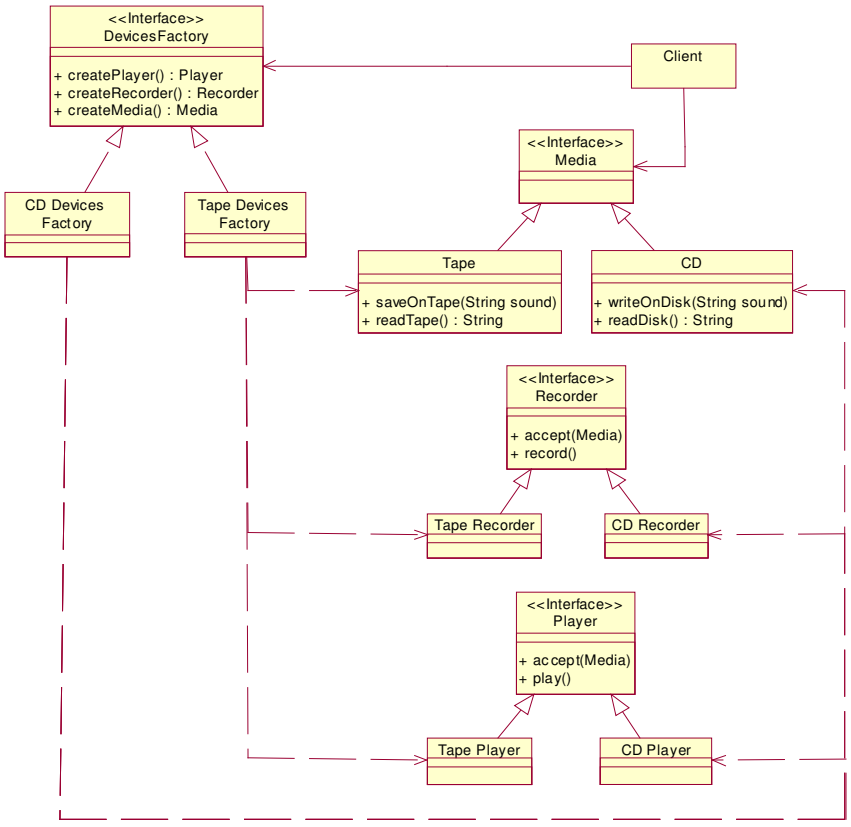
Il pattern "*Abstract Factory*" si basa sulla creazione di interfacce per ogni tipo di prodotto. Ci saranno poi concreti prodotti che implementano queste interfacce, stesse che consentiranno ai clienti di fare uso dei prodotti. Le famiglie di prodotti saranno create da un oggetto noto come *factory*. Ogni famiglia avrà una particolare *factory* che sarà utilizzata dal Cliente per creare le istanze dei prodotti. Siccome non si vuole legare al Cliente un tipo specifico di *factory* da utilizzare, le *factory* implementeranno una interfaccia comune che sarà dalla conoscenza del Cliente.

1.4. Struttura del pattern



1.5. Applicazione del pattern

Schema del modello



Partecipanti

- **AbstractFactory**: interfaccia DevicesFactory.
  - Dichiara una interfaccia per le operazioni che creano e restituiscono i prodotti.
  - Nella dichiarazione di ogni metodo, i prodotti restituiti sono dei tipi AbstractProduct.
- **ConcreteFactory**: classi TapeDevicesFactory **e** CDDevicesFactory**.**
  - Implementa l'AbstractFactory, fornendo le operazioni che creano e restituiscono oggetti corrispondenti a prodotti specifici (**ConcreteProduct**).
- **AbstractProduct**: interfacce Media, Recorder e Player.
  - Dichiarano le operazioni che caratterizzano i diversi tipi generici di prodotti.
- **ConcreteProduct**: classi Tape, TapeRecorder, TapePlayer, CD, CDRecorder e CDPlayer.
  - Definiscono i prodotti creati da ogni ConcreteFactory.
- **Client**: classe Client.
  - Utilizza l'**AbstractFactory** per rivolgersi alla **ConcreteFactory** di una famiglia di prodotti.
  - Utilizza i prodotti tramite la loro interfaccia **AbstractProduct**.

Descrizione del codice

Si creano le interfacce delle classi che devono implementare i prodotti di tutte le famiglie. `Media` è una *marker interface* che serve per identificare i supporti di registrazione, intanto le interfacce `Player` e `Recorder` dichiarano i metodi che i clienti invocheranno nei riproduttori e registratori.

```
public interface Media { }

public interface Player {
    public void accept( Media med );
    public void play( );
}

public interface Recorder {
    public void accept( Media med );
    public void record( String sound );
}
```

S'implementano le concrete classi che costituiscono i prodotti delle famiglie.

Le classi appartenenti alla famiglia di prodotti basati sul nastro sono `Tape`, `TapeRecorder` e `TapePlayer`:

```
public class Tape implements Media {

   private String tape= "";
```

```
  public void saveOnTape( String sound ) {
       tape = sound;
   }

   public String readTape( ) {
       return tape;
   }

}
```

```
public class TapeRecorder implements Recorder {

    Tape tapeInside;

    public void accept( Media med ) {
        tapeInside = (Tape) med;
    }

    public void record( String sound ) {
        if( tapeInside == null )
         System.out.println( "Error: Insert a tape." );
        else
           tapeInside.saveOnTape( sound );
    }

}
```

```
public class TapePlayer implements Player {

    Tape tapeInside;

    public void accept( Media med ) {
        tapeInside =  (Tape) med;
    }

    public void play( ) {
        if( tapeInside == null )
         System.out.println( "Error: Insert a tape." );
        else
           System.out.println( tapeInside.readTape() );
    }

}
```

Le classi appartenenti alla famiglia di prodotti basati sul compact disc sono `CD`, `CDPlayer` e `CDRecorder`:

```
public class CD implements Media{

   private String track = "";

   public void writeOnDisk( String sound ) {
       track = sound;
   }

   public String readDisk( ) {
       return track;
   }

}
```

```
public class CDRecorder implements Recorder {

    CD cDInside;

    public void accept( Media med ) {
        cDInside = (CD) med;
    }
```

```
    public void record( String sound ) {
        if( cDInside == null )
         System.out.println( "Error: No CD." );
        else
           cDInside.writeOnDisk( sound );
    }

}
```

```
public class CDPlayer implements Player {

    CD cDInside;

    public void accept( Media med ) {
        cDInside = (CD) med;
    }

    public void play( ) {
        if( cDInside == null )
         System.out.println( "Error: No CD." );
        else
           System.out.println( cDInside.readDisk() );
    }

}
```

L'interfaccia `DevicesFactory` specifica la firma dei metodi che restituiscono gli oggetti di qualunque tipo di famiglia.

```
public interface DevicesFactory {

   public Player createPlayer();
   public Recorder createRecorder();
   public Media createMedia();

}
```

Si implementano le **ConcreteFactory**: `TapeDevicesFactory`, per la famiglia di prodotti basata sul nastro, e `CDDevicesFactory`, per quelli basati sul compact disc.

```
public class TapeDevicesFactory implements DevicesFactory {

   public Player createPlayer() {
      return new TapePlayer();
   }

   public Recorder createRecorder() {
      return new TapeRecorder();
   }

   public Media createMedia() {
      return new Tape();
   }

}
```

```
public class CDDevicesFactory implements DevicesFactory {

   public Player createPlayer() {
      return new CDPlayer();
   }
   public Recorder createRecorder() {
      return new CDRecorder();
   }
   public Media createMedia() {
      return new CD();
   }

}
```

La classe `Client` definisce gli oggetti che utilizzano i prodotti d'ogni famiglia. Il metodo `selectTechnology` riceve un oggetto corrispondente ad una famiglia particolare e lo registra dentro i propri attributi. Il metodo `test` crea una istanza d'ogni particolare tipo di prodotto e applica i diversi metodi forniti dalle interfacce che implementano i prodotti. Si deve notare che il cliente utilizza i prodotti, senza avere conoscenza di quali sono concretamente questi.

```
class Client {

   DevicesFactory technology;

   public void selectTechnology( DevicesFactory df ) {
      technology = df;
   }

   public void test(String song) {

      Media    media    = technology.createMedia();
      Recorder recorder = technology.createRecorder();
      Player   player   = technology.createPlayer();

      recorder.accept( media );
      System.out.println( "Recording the song  : " + song );
      recorder.record( song );
      System.out.println( "Listening the record:" );
      player.accept( media );
      player.play();
   }

}
```

Finalmente, si presenta il programma che crea una istanza di un oggetto `Client`, il quale riceve tramite il metodo `selectTechnology` una istanza di una **ConcreteFactory**. In questo esempio particolare, si assegna una prova ad ogni famiglia di prodotti:

```
public class AbstractFactoryExample {

   public static void main ( String[] arg ) {

      Client client = new Client();

      System.out.println( "**Testing tape devices" );
      client.selectTechnology( new TapeDevicesFactory() );
      client.test( "I wanna hold your hand..." );

      System.out.println( "**Testing CD devices" );
      client.selectTechnology( new CDDevicesFactory() );
      client.test( "Fly me to the moon..." );

   }

}
```

Osservazioni sull'esempio

Nell'esempio si è voluto accennare la stretta interazione tra le classi costituenti ogni famiglia di prodotto, in modo tale di costringere all'utilizzo, nelle prove, di prodotti dello stesso tipo. In ogni particolare famiglia le interfacce tra i componenti sono diverse, e non conosciute dal cliente. In particolare, si noti che l'interfaccia d'ogni tipo di Media verso i corrispondenti registratori e riproduttori varia da una famiglia all'altra.

Esecuzione dell'esempio

```
C:\Patterns\Creational\Abstract Factory\>java AbstractFactoryExample

**Testing tape devices
Recording the song  : I wanna hold your hand...
Listening the record:
I wanna hold your hand...

**Testing CD devices
Recording the song  : Fly me to the moon...
Listening the record:
Fly me to the moon...
```

1.6. Osservazioni sull'implementazione in Java

Dovuto al fatto che né l'**AbstractFactory** né gli **AbstractProduct** implementano operazioni, in Java diventa più adeguato codificarli come interfacce piuttosto che come classi astratte.