21. Strategy

(GoF pag. 315)

21.1. Descrizione

Consente la definizione di una famiglia d'algoritmi, incapsula ognuno e gli fa intercambiabili fra di loro. Questo permette modificare gli algoritmi in modo indipendente dai clienti che fanno uso di essi.

21.2. Esempio

La progettazione di una applicazione che offre delle funzionalità matematiche, considera la gestione di una apposita classe (MyArray) per la rappresentazione di vettori di numeri. Tra i metodi di questa classe si ha definito uno che esegue la propria stampa. Questo metodo potrebbe stampare il vettore nel seguente modo (chiamato, ad es. MathFormat):

```
{ 12, -7, 3, … }
```

oppure di questo altro modo (chiamato, ad. es. SandardFormat):

```
Arr[0]=12 Arr[1]=-7 Arr[2]=3 …
```

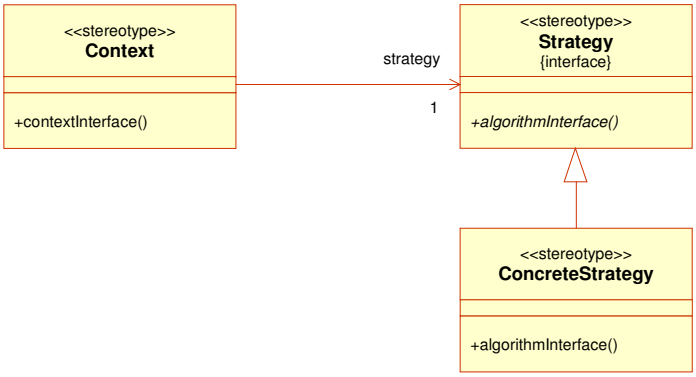
E' anche valido pensare che questi formati potrebbero posteriormente essere sostituiti da altri.

Il problema è trovare un modo di isolare l'algoritmo che formatta e stampa il contenuto dell'array, per farlo variare in modo indipendente dal resto dell'implementazione della classe.

21.3. Descrizione della soluzione offerta dal pattern

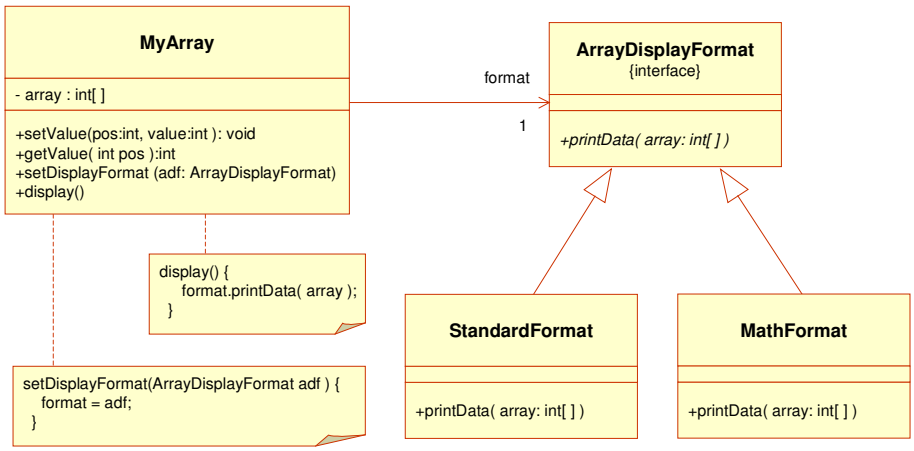
Lo "*Strategy*" pattern suggerisce l'incapsulazione della logica di ogni particolare algoritmo, in apposite classi (ConcreteStrategy) che implementano l'interfaccia che consente agli oggetti MyArray (Context) di interagire con loro. Questa interfaccia deve fornire un accesso efficiente ai dati del Context, richiesti da ogni ConcreteStrategy, e viceversa.

21.4. Struttura del Pattern



21.5. Applicazione del pattern

Schema del modello



Partecipanti

- **Strategy**: interfaccia ArrayDisplayFormat.
  - Dichiara una interfaccia comune per tutti gli algoritmi supportati. Il **Context** utilizza questa interfaccia per invocare gli algoritmi definiti in ogni **ConcreteStrategy**.
- **ConcreteStrategy**: classi StandardFormat e MathFormat.
  - Implementano gli algoritmi che usano la interfaccia **Strategy**.
- **Context**: classe MyArray.
  - Viene configurato con un oggetto **ConcreteStrategy** e mantiene un riferimento verso esso.
  - Può specificare una interfaccia che consenta alle **Strategy** accedere ai propri dati.

Descrizione del codice

Si implementa la classe `MyArray` (**Context**) che mantiene al suo interno un array di numeri, gestiti tramite i metodi `setValue` e `getValue`. La particolare modalità di stampa rimane a carico di oggetti che implementano l'interfaccia `ArrayDisplayFormat`. Il particolare oggetto che incapsula la procedura di stampa scelta, viene settato tramite il metodo `setDisplayFormat`, intanto che la procedura stessa di stampa viene invocata tramite il metodo `display`:

```
public class MyArray {

  private int[] array;
  private int size;
  ArrayDisplayFormat format;


  public MyArray( int size ) {
     array = new int[ size ];
  }

  public void setValue( int pos, int value ) {
     array[pos] = value;
  }

  public int getValue( int pos ) {
    return array[pos];
  }

  public int getLength( int pos ) {
    return array.length;
  }

  public void setDisplayFormat( ArrayDisplayFormat adf ) {
    format = adf;
  }

  public void display() {
     format.printData( array );
  }

}
```

Si specifica l'interfaccia `ArrayDisplayFormat`, da implementare in ogni classe fornitrice dell'operazione di stampa.

```
public interface ArrayDisplayFormat {

  public void printData( int[] arr );

}
```

Le strategie di stampa sono implementate nelle classi `StandardFormat` (per il formato "{ a, b, c, … }") e `MathFormat` (per il formato "Arr[0]=a Arr[1]=b Arr[2]=c …"):

```
public class StandardFormat implements ArrayDisplayFormat {

   public void printData( int[] arr ) {
      System.out.print( "{ " );
      for(int i=0; i < arr.length-1 ; i++ )
         System.out.print( arr[i] + ", " );
      System.out.println( arr[arr.length-1] + " }" );

   }
}
```

```
public class MathFormat implements ArrayDisplayFormat {

   public void printData( int[] arr ) {
      for(int i=0; i < arr.length ; i++ )
         System.out.println( "Arr[ " + i + " ] = " + arr[i] );
   }
}
```

Finalmente si presenta il codice che fa uso della classe `MyArray`. Si noti che è questo chi sceglie e istanza la particolare strategia di presentazione dei dati.

```
public class StrategyExample {

 public static void main (String[] arg) {

   MyArray m = new MyArray( 10 );
   m.setValue( 1 , 6 );
   m.setValue( 0 , 8 );
   m.setValue( 4 , 1 );
   m.setValue( 9 , 7 );
   System.out.println("This is the array in 'standard' format");
   m.setDisplayFormat( new StandardFormat() );
   m.display();
   System.out.println("This is the array in 'math' format:");
   m.setDisplayFormat( new MathFormat() );
   m.display();
 }

}
```

Osservazioni sull'esempio

I nomi forniti alle tipologie di stampa sono stati inventati per questo esempio.

Esecuzione dell'esempio

```
C: \Design Patterns\Behavioral\Strategy>java StrategyExample

This is the array in 'standard' format :
{ 8, 6, 0, 0, 1, 0, 0, 0, 0, 7 }

This is the array in 'math' format:
Arr[ 0 ] = 8
Arr[ 1 ] = 6
Arr[ 2 ] = 0
Arr[ 3 ] = 0
Arr[ 4 ] = 1
Arr[ 5 ] = 0
Arr[ 6 ] = 0
Arr[ 7 ] = 0
Arr[ 8 ] = 0
Arr[ 9 ] = 7
```

21.6. Osservazioni sull'implementazione in Java

Non ci sono aspetti particolari da tenere in conto.