

Intento

Definire lo scheletro di un algoritmo, rimandando alcuni passi alle subclass. Template consente alle subclass di ridefinire alcune parti di un algoritmo, senza per questo cambiarne la struttura d'insieme.

Motivazione

Si supponga di ideare un framework di classi su cui poi basare delle implementazioni concrete. Un metodo di una di queste classi in generale si occupa di implementare un algoritmo. Essendo un framework, è plausibile che a questo livello si possa disporre della struttura di un algoritmo, però non tutta implementabile. Si desidera lasciare che siano le subclass a definire i passi che mancano dell'algoritmo. Utilizzando Template questo è semplice, si definisce un MetodoTemplate() che implementa la struttura dell'algoritmo a livello di framework ovvero in una classe astratta:

```
procedure ClasseAstratta.MetodoTemplate;  
begin  
    // Passi a piacere ...  
    OperazionePrimitiva1; // metodo astratto  
    // altro ...  
    OperazionePrimitiva2; // metodo astratto  
    // ... ed operazioni a piacere.  
end;
```

Le subclass di *ClasseAstratta* che utilizzano MetodoTemplate devono solo implementare *OperazionePrimitiva1* e *OperazionePrimitiva2*.

La motivazione consiste nella necessità di definire un algoritmo a livello della sua struttura e di alcuni particolari, lasciando la possibilità alle subclass di implementare alcune funzioni o parti di tale algoritmo (quelle di cui non è possibile dare una definizione all'atto della codifica della classe astratta).

Applicabilità

1. Da utilizzare per implementare le parti invarianti di un algoritmo, lasciando alle sottoclassi la responsabilità di implementare i comportamenti che possono variare.
2. Per riunire in un solo punto il codice comune a più sottoclassi, in modo da localizzarlo. Le differenze vanno poi implementate da ogni sottoclasse.
3. Per controllare l'estensibilità delle sottoclassi: si definisce un metodo Template che richiama dei metodi "vuoti" (**hook**) in precisi e predeterminati punti, in modo da consentire alle sottoclassi di personalizzare un algoritmo solo in tali predefiniti punti.

Struttura

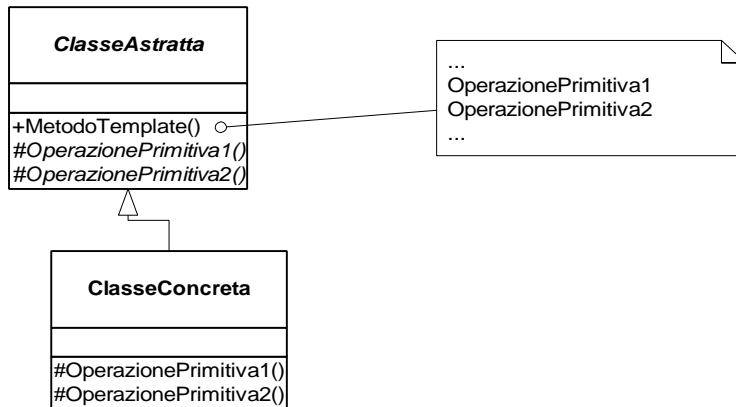


Fig.1 - Diagramma di classe del pattern Template

Conseguenze

I metodi di tipo template sono uno strumento importante per organizzare e riunire il comportamento comune nelle classi delle librerie, concentrandolo in una classe base. Un metodo template può definire ovvero utilizzare al suo interno dei metodi astratti oppure dei metodi cosiddetti **hook**. Un metodo astratto deve essere concretizzato dalle subclass. Un metodo **hook** è costituito da un metodo concreto che esegue alcuna azione di default, in pratica è un segnaposto con un comportamento di default per le variazioni di un algoritmo che verranno eventualmente concretizzate in una subclass. Un metodo che realizza il pattern template può utilizzare sia metodi astratti che **hook**, per questo è importante che siano facilmente riconoscibili, perché dei primi di deve fare l'override, dei secondi solo se necessario.

Implementazione

Per identificare facilmente i metodi di cui occorre fare l'override, è convenzione di alcune librerie (MacApp) di anteporre al nome del metodo il prefisso Do, p. es.: `DoPrimitiveOperation1`.

Codice di esempio

Questo è un esempio in Delphi in cui si applica il pattern Template:

```
...
type
{** Classe base con metodi astratti, che impostano l'algoritmo ma non
lo realizzano; verrà implementato dalla classe figlia }
TPLibTemplateAbstractClass=class(TObject)
protected
    procedure DoPrimitiveOperation1; virtual; abstract;
    procedure DoPrimitiveOperation2; virtual; abstract;
public
    {** Questo è il metodo che contiene lo scheletro dell'algoritmo, ma che
    non lo implementa completamente }
    procedure TemplateMethod; virtual;
end;

{** Prima classe concreta a cui è richiesta l'implementazione dei metodi
astratti che TemplateMethod richiama }
TPLibTemplateConcreteClass1=class(TPLibTemplateAbstractClass)
protected
    {** Implementazione delle operazioni del TemplateMethod }
    procedure DoPrimitiveOperation1; override;
    procedure DoPrimitiveOperation2; override;
end;
```

```

    {** Seconda classe concreta a cui è richiesta l'implementazione dei metodi
    astratti che TemplateMethod richiama }
    TPLibTemplateConcreteClass2=class(TPLibTemplateAbstractClass)
    protected
        {** Implementazione delle operazioni del TemplateMethod }
        procedure DoPrimitiveOperation1; override;
        procedure DoPrimitiveOperation2; override;
    end;

...
implementation
...
{** TPLibTemplateAbstractClass }
procedure TPLibTemplateAbstractClass.TemplateMethod;
begin
    // codice a piacere ...
    showmessage('Prima parte comune dell''algoritmo, definita nella classe base');
    // parte astratta
    DoPrimitiveOperation1;
    DoPrimitiveOperation2;
    // altro codice a piacere ...
    showmessage('Seconda parte comune dell''algoritmo, definita nella classe base');
end;

{** TPLibTemplateConcreteClass }
procedure TPLibTemplateConcreteClass1.DoPrimitiveOperation1;
begin
    {** il codice che implementa veramente il TemplateMethod va qui}
    showmessage('Operazione 1');
end;

procedure TPLibTemplateConcreteClass1.DoPrimitiveOperation2;
begin
    {** il codice che implementa veramente il TemplateMethod va qui}
    showmessage('Operazione 2');
end;

{ TPLibTemplateConcreteClass2 }
procedure TPLibTemplateConcreteClass2.DoPrimitiveOperation1;
begin
    showmessage('Operazione 1 della seconda classe concreta');
end;

procedure TPLibTemplateConcreteClass2.DoPrimitiveOperation2;
begin
    showmessage('Operazione 2 della seconda classe concreta');
end;
...

```

Per l'esempio completo vedere file Template10.zip.

Usi nella VCL

- in sospenso -

Pattern correlati

Factory: DP107

Strategy: DP 315

Riferimenti

Autore codice e documentazione: Marco Planchestainer <m.plank@usa.net>

Versione: 1.0 del 31/01/1999

Sito Internet: <http://members.tripod.com/mplank>, rif. ProjectOO