

## Esercizio sul pattern “decorator”

Il software di gestione di una catena di bar specializzati nel servire caffè espresso di vari tipi deve modellare tre differenti specie di caffè (*arabica*, *gentile*, *decaf*) ognuna delle quali può essere servita con nessuna, una o due modifiche a scelta (*con\_latte*, *con\_cacao*). Nel caso di richiesta di due modifiche queste devono essere diverse tra loro. Il costo di un caffè espresso è determinato dal costo della specie usata per prepararlo più l'eventuale costo di tutte le modifiche che il cliente richiede.

Un'implementazione proposta per tale software suggerisce di implementare una classe *deferred* per rappresentare il caffè generico con una feature *deferred* per calcolare il costo del caffè, una serie di features *effective* che tengono traccia del costo base di ogni specie e di ogni modifica. Si veda qua sotto una traccia di tale implementazione.

### deferred class

CAFFE

### feature {ANY}

costo : REAL

-- calcola il costo del caffè con le aggiunte desiderate.

deferred

end

costo\_arabica : REAL = 2.1

-- il costo base di un espresso della specie arabica

costo\_gentile : REAL = 1.7

-- il costo base di un espresso della specie gentile

costo\_decaf : REAL = 1.9

-- il costo base di un espresso della specie decaf

costo\_latte : REAL = 0.1

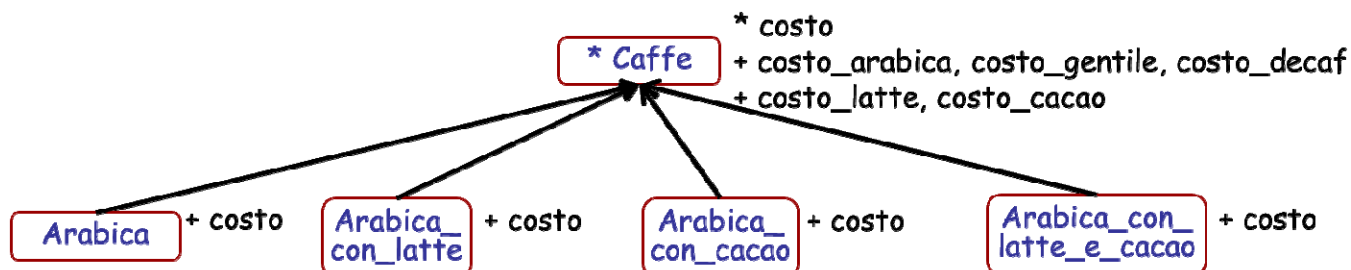
-- il costo base di una modifica con aggiunta di latte

costo\_cacao : REAL = 0.2

-- il costo base di una modifica con aggiunta di cacao

end

Inoltre, la proposta suggerisce di avere una sottoclasse per tutte le possibili combinazioni di specie e modifiche. Ognuna di tali sottoclassi rende *effective* la feature per il calcolo del costo del caffè. Si veda qua sotto un frammento della struttura di classi che deriverebbe da questa proposta.



Il sistema proposto avrebbe come *root* la classe *SERVI\_CAFFE* sotto descritta, la cui *root procedure* è *servi*. La scelta da parte dell'utente della specie “nessuna” viene interpretato come richiesta di terminare l'esecuzione del sistema. La scelta come prima modifica di “nessuna” viene interpretata come richiesta dell'utente di non aggiungere niente.

```

class
  SERVI_CAFFE

feature {NONE}
  caffe_da_servire : CAFFE

feature {ANY}
  servi
    local
      specie_scelta, modifica_1, modifica_2 : STRING
      somma_da_incassare : REAL
    do
      from
        -- "acquisizione dall'utente dei valori di specie_scelta, modifica_1, modifica_2"
      until
        specie_scelta ~ "nessuna"
      loop
        if specie_scelta ~ "arabica" then
          if modifica_1 ~ "nessuna" then
            caffe_da_servire := create {ARABICA}
          elseif modifica_1 ~ "latte" then
            if modifica_2 ~ "nessuna" then
              caffe_da_servire := create {ARABICA_CON_LATTE}
            else -- qui modifica_2 vale "cacao"
              caffe_da_servire := create {ARABICA_CON_LATTE_E_CACAO}
            end
          else -- qui modifica_1 vale "cacao"
            if modifica_2 ~ "nessuna" then
              caffe_da_servire := create {ARABICA_CON_CACAO}
            else -- qui modifica_2 vale "latte"
              caffe_da_servire := create {ARABICA_CON_LATTE_E_CACAO}
            end
          end
        elseif specie_scelta ~ "gentile" then
          "continua con lo stesso tipo di codice del caso dell'arabica"
        end
        else -- qui specie_scelta vale "decaf"
          "continua con lo stesso tipo di codice del caso dell'arabica"
        end
      end
      somma_da_incassare := caffe_da_servire.costo
      -- "acquisizione dall'utente dei valori di specie_scelta, modifica_1, modifica_2"
    end
  end
end
end

```

Con l'implementazione proposta si otterrebbero 12 classi (oltre a *SERVI\_CAFFE*), cioè 4 classi per ognuna delle 3 possibili specie di caffè.

Le implementazioni della feature *costo* per le 4 classi del frammento proposto sono le seguenti:

```

class
  ARABICA
inherit
  CAFFE
feature {ANY}
  costo : REAL
do
  Result := costo_arabica
end
end

```

```
class
  ARABICA_CON_LATTE
inherit
  CAFFE
feature {ANY}
  costo : REAL
do
  Result := costo_arabica + costo_latte
end
end

class
  ARABICA_CON_CACAO
inherit
  CAFFE
feature {ANY}
  costo : REAL
do
  Result := costo_arabica + costo_cacao
end
end

class
  ARABICA_CON_LATTE_E_CACAO
inherit
  CAFFE
feature {ANY}
  costo : REAL
do
  Result := costo_arabica + costo_latte + costo_cacao
end
end
```

I problemi di questa implementazione sono essenzialmente legati alla difficoltà di manutenzione:

- il codice viene replicato inutilmente perché la gestione delle modifiche viene replicata per ognuna delle specie
- in caso di rimozione di una modifica vanno modificate tutte le classi in cui è presente la modifica
- in caso di inserimento di una modifica vanno modificate le classi di tutte le specie.

Per ottenere un migliore riuso del codice ed una migliore manutenibilità va separata l'implementazione relativa alla specie di caffè da quella relativa alle modifiche. Inoltre, le due implementazioni vanno rese componibili.

Questo si può ottenere evitando di dichiarare le classi relative alle specie con tutte le possibili modifiche e mantenendo le sole classi relative alle specie base. L'implementazione della classi per la specie *ARABICA* è uguale alla precedente implementazione e quelle per le altre specie si derivano per analogia.

L'implementazione relativa alle modifiche si ottiene definendo una classe per ognuno dei singoli tipi di modifica. Per rendere tali classi che modellano le modifiche componibili con le classi che modellano le specie di caffè, le classi relative alle modifiche devono essere anch'esse discendenti dalla superclasse comune a tutte le specie di caffè. Ognuna di esse dovrà rendere *effective* la *feature* di costo in modo da aggiungere al costo di partenza quello relativo all'aggiunta della propria modifica. La componibilità delle modifiche tra loro si ottiene definendo tali classi relative alle modifiche come sottoclassi di una stessa classe *deferred* che contiene il meccanismo base di componibilità, basato sul definire un certo tipo di caffè come il caffè di base, indipendentemente dal fatto che siano presenti o meno delle modifiche.

Questo schema di soluzione, il cui dettaglio viene discusso nel seguito è uno dei pattern importanti nella realizzazione del software ed è chiamato "*decorator*" (decoratore) perché aggiunge una "decorazione" a qualcosa che già esiste, mediante un meccanismo che permette di comporre più decorazioni una sull'altra. L'implementazione nel seguito contribuisce a chiarire l'approccio.

```

deferred class
  CAFFE_SERVITO
inherit
  CAFFE
feature {NONE}
  caffe_base : CAFFE
feature {ANY}
  set_base (base : CAFFE)
  do
    caffe_base  $\equiv$  base
  end
end

```

A questo punto, per ognuna delle singole modifiche si definisce una specifica sottoclasse di *CAFFE\_SERVITO* che implementa la feature *costo* che è rimasta *deferred* nella catena di ereditarietà a partire da *CAFFE* in modo da tener presente nel calcolo del costo la specifica modifica modellata dalla classe.

Qui sotto è presente la classe nel caso di modifica *CAFFE\_PIU\_LATTE* e per estensione si possono derivare le implementazioni per le altre aggiunte.

```

class
  CAFFE_PIU_LATTE
inherit
  CAFFE_SERVITO
create
  set_base
feature {ANY}
  costo : REAL
  do
    Result := caffe_base.costo + costo_latte
  end
end

```

Per gestire nello stesso modo il caso di caffè senza modifiche conviene anche aggiungere una classe che modella con lo stesso meccanismo il caso di nessuna modifica, implementata come segue:

```

class
  CAFFE_PIU_NIENTE
inherit
  CAFFE_SERVITO
create
  set_base
feature {ANY}
  costo : REAL
  do
    Result := caffe_base.costo
  end
end

```

Le classi così definite possono essere composte all'atto della creazione delle istanze, in modo che una nuova istanza prenda come base una precedente istanza ed aggiunga ad essa il suo specifico comportamento. Così facendo, l'invocazione della feature *costo* causerà l'esecuzione in successione delle relative versioni della stessa feature di ognuna delle istanze che sono state composte.

La *root class* e la *root procedure* del sistema diventano quindi le seguenti:

```

class
  SERVI_CAFFE

feature {NONE}
  caffe_da_servire : CAFFE
  specie_scelta, modifica_1, modifica_2 : STRING

feature {NONE}
  aggiungi (modifica : STRING; caffe_base: CAFFE) : CAFFE
  do
    if modifica ~ "nessuna" then
      Result = create {CAFFE_PIU_NIENTE}. set_base (caffe_base)
    elseif modifica ~ "latte" then
      Result = create {CAFFE_PIU_LATTE}. set_base (caffe_base)
    else -- qui modifica vale "cacao"
      Result = create {CAFFE_PIU_CACAO}. set_base (caffe_base)
    end
  end

feature {ANY}
  servi
  local
    somma_da_incassare : REAL
  do
    from
      -- "acquisizione dall'utente dei valori di specie_scelta, modifica_1, modifica_2"
    until
      specie_scelta ~ "nessuna"
    loop
      if specie_scelta ~ "arabica" then
        caffe_da_servire = create {ARABICA}
      elseif specie_scelta ~ "gentile" then
        caffe_da_servire = create {GENTILE}
      else -- qui specie_scelta vale "decaf"
        caffe_da_servire = create {DECAF}
      end
      if modifica_1 /~ "nessuna" then
        caffe_da_servire = aggiungi (modifica_1, caffe_da_servire)
        caffe_da_servire = aggiungi (modifica_2, caffe_da_servire)
      end
      somma_da_incassare = caffe_da_servire. costo
      -- "acquisizione dall'utente dei valori di specie_scelta, modifica_1, modifica_2"
    end
  end
end

```

In questa implementazione il codice è riusato meglio dal momento che le aggiunte vengono gestite in un'unica procedura (la nuova feature *aggiungi*).

Inoltre la manutenzione è facilitata: nel caso di variazioni sulle specie di caffè si deve intervenire solo sulla classe variata o definendo una nuova classe (se viene introdotta una nuova specie di caffè). Anche nel caso di variazioni sulle modifiche si deve intervenire solo sulla procedura *aggiungi* e/o definire una nuova classe (se viene introdotta una nuova modifica).

Nei progetti realizzati vi è la versione base, senza utilizzo del pattern "decorator", la versione col pattern "decorator" (obbligando la scelta di due differenti modifiche) ed una sua variazione che consente un'ulteriore modifica ("grappa") e la replica della prima modifica e descrive alla fine il caffè servito.