Conception Object

La strategie

Alain Plantec

UBO

13 novembre 2012

Gestion du choix d'une action

Souvent, nos objects sont paramétrés pour effectuer telle ou telle action suivant le contexte. L'alternative peut être basée sur la valeur d'un paramètre d'une méthode ; On a alors le type de code suivant :

```
doAction : flag
    flag = case1
        ifTrue : [self actionForCase1]
        ifFalse : [flag = case2
            ifTrue : [self actionForCase2]
            ifFalse : [self actionForCase3].
```

Gestion du choix d'une action

Le choix paut aussi être basé sur la valeur d'une variable d'instance. On a alors le type de code suivant :

```
doAction
    self flag = case1
        ifTrue : [self actionForCase1]
        ifFalse : [self flag = case2
            ifTrue : [self actionForCase2]
            ifFalse : [self actionForCase3].
```

La valeur du flag est alors passée lors de la construction de l'objet et/ou modifiée par un accesseur.

Problème

Ce code n'est pas évolutif sans modification de la classe qui met en oeuvre le choix.
Si je veux ajouter un quatrième cas, je doit modifier *doAction* et je doit ajouter *actionForCase*4.
On ne respecte donc pas l'OCP !

```
doAction
    self flag = case1
        ifTrue : [self actionForCase1]
        ifFalse : [self flag = case2
            ifTrue : [self actionForCase2]
            ifFalse : [self flag = case3
                ifTrue : [self actionForCase3]
                ifFalse : [self actionForCase4]
```

Brainstorming : comment mettre en oeuvre le choix et ajouter de nouveaux choix sans devoir modifier la classe ?

Composition et délégation

Pour résoudre ce problème de conception, on peut représenter l'alternative de choix d'algorithme (cas1, cas2, ...) non plus par la valeur d'un flag mais par un objet pouvant lui même exécuter l'action.

- On a une class abstraite pour les cas.
- Cett classe représente les différentes **stratégies** possibles pour l'action.
- Chacune de ces sous-classes :
  - représente un cas possible.
  - met en oeuvre une fonction membre pour l'action
- La classe qui comportait un flag est maintenant associé à une stratégie.
- l'exécution de l'action est délégée à la stratégie

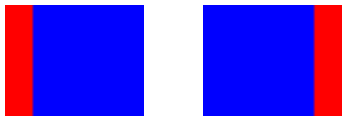
```
doAction
   self caseStrategy doAction
```

Une barre dans un morph

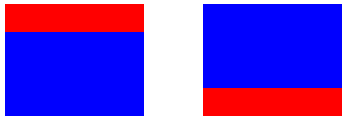
Objectif

L'objectif est de construire un rectangle bleu contenant un rectangle rouge.
Illustrations :

- en haut ou en bas, il prend toute la largeur



- à gauche ou à droite, il prend toute la hauteur



- pas de barre rouge

- mais aussi coin haut gauche, coin bas droit etc...

Quelle(s) solution(s) pour paramétrer le placement de la barre rouge dans le rectangle bleu ?

Une barre dans un morph

Deux tests à faire fonctionner

On Programme la classe *MorphWithBar* (spécialisation de *Morph*).

```
Morph subclass : #MorphWithBar
    instanceVariableNames : ''
    classVariableNames : ''
    poolDictionaries : ''
    category : 'CO'
```

Test avec un constructeur :

```
MorphWithBar barOnTheTop openInWorld.
```

Test qui utilise une méthode d'instance pour forcer le placement :

```
bwr  := MorphWithBar new.
bwr barOnTheTop.
bwr openInWorld.
```

Une barre dans un morph

Note sur l'utilisation d'un LayoutFrame pour le placement

Pour placer un morph dans un autre, on peut utiliser un LayoutFrame.
Exemple avec la barre en haut :

```
| m |
m  := Morph new layoutPolicy : ProportionalLayout new.
sm  := Morph new color : Color red.
m addMorph : sm fullFrame :
   (LayoutFrame
      fractions : (0 @ 0 corner : 1 @ 0)
      offsets : (0 @ 0 corner : 0 @ 10)).
m openInWorld.
```

Pour en savoir plus sur le placement d'un morph dans un autre, regarder sur le site collaboratif de l'**Ofset** (http ://community.ofset.org/).