Principe de l’encapsulation

Chapitre 4

Qu’est-ce l’encapsulation ?

L’encapsulation est un principe fondamental à la programmation orientée objet. Il permet de garantir le bon usage d’objets en limitant les manipulations possibles de l’état d’un objet.

Ce contrôle donne des objets qui ne peuvent jamais être dans un état incohérente et suivent toujours les comportements attendus par l’auteur d’une classe.

C’est la **solution au problème de validation** vu dans la dernière capsule.

Attribus privés

Dorénavant, tous les attributs devront être **private** au lieu de public. Ceci rendra inaccessible les attributs à un client, c'est-à-dire que tout code à l'extérieur de la classe ne peut pas accéder (lire) et modifier l'état d'un objet.

```
class Rectangle
{
    private int largeur;
    private int hauteur;

    ...
}
```

```
class Program
{
    Rectangle r = ...
    //erreur
    r.largeur = -999;
}
```

Accesseur et mutateur

Les accesseurs et mutateurs sont des méthodes qui implémente une le principe de l'encapsulation. Car les attributs sont cachés des clients, ces méthodes permettent de contrôler avec précision l'accès et la modification d'un attribut.

Par exemple, il est possible de forcer un client à utiliser un mutateur s'il veut modifier un attribut d'un objet. Un mutateur est une méthode, alors la méthode peut contenir de la validation. Ceci règle le problème de validation et garanti un état cohérent !

Exemple

```
class                                   Rectangle
{
                    private     int     largeur;
    private int hauteur;

    public void SetLargeur(int nouvelleValeur)
                                                {
                    if   (nouvelleValeur   <   1)
                          nouvelleValeur   =   1;
                    largeur   =   nouvelleValeur;
                                                }
}
```

```
class                                   Progra
{
     Rectangle r = new Rectangle()
          /*   forcer   à   utilise
        un   mutateur   pour   change
                la    largeur    *
                r.SetLargeur(-999)

       /*  la  largeur  du  rectangl
       est  1.  autocorrection  !  *
}
```

Syntaxe des accesseurs et mutateurs

Accesseurs :

```
public typeAttribut GetNomAttribut()
{
    return attribut;
}
```

Mutateurs :

```
public/private void SetNomAttribut(typeAttribut nouvelleValeur)
{
    //validation si nécessaire suivi de :
    attribut = nouvelleValeur;
}
```

Changements dans le constructeur

Au lieu d’affecter directement les attributs dans le constructeur de la classe, la bonne pratique est d’utiliser les mutateurs.

```
public Rectangle()
{
    SetLargeur(3);
    SetHauteur(4);
}
```

Règles

1. Toujours utiliser les accesseurs et mutateurs au lieu des attributs directements;

2. Un accesseur doit seulement retourner **un** attribut (pas d'effets de bord);

3. Un mutateur doit seulement modifier **un** attribut (pas d'effets de bord);

4. Privilégié les mutateurs privés pour avoir des objets plus simples (plus de limites → plus facile à utiliser).