Quelques notions utiles aux tests

LE COUPLAGE

Définition

- Le couplage entre deux composants est un indicateur du volume d’informations qu’ils communiquent entre eux.
- On parle de couplage « fort » ou « serré » entre deux composants s’ils échangent beaucoup de données.
- Au contraire, in parle de couplage faible si les composants sont indépendants ou s’ils échangent un minimum de données.

Les niveaux de couplage

Selon ~~Wikipedia~~ Pressman[1], il existe sept niveaux de couplage, du plus faible au plus fort :

1. **Sans couplage** : pas d’échange d'information.
2. **Par données** : échanges de l'information par des méthodes utilisant des arguments (paramètres) de type simple (nombre, chaîne de caractères, tableau).
3. **Par paquet** : les composants échangent de l'information par des méthodes utilisant des arguments de type composé (structure, classe).

Les niveaux de couplage

4. **Par contrôle** : les composants se passent ou modifient leur contrôle par changement d'un drapeau (verrou).
5. **Externe** : les composants échangent de l'information par un moyen de communication externe (fichier, pipeline, lien de communication).
6. **Commun (global)** : les composants échangent de l'information via un ensemble de données (variables) commun.
7. **Par contenu (interne)** : les composants échangent de l'information en lisant et écrivant directement dans leurs espaces de données (variables) respectifs.

Problèmes d’un couplage fort

- Plus le couplage entre deux objets est fort, plus la modification de l’un entraîne la modification de l’autre, et plus les objets sont difficilement testables.
- Il est difficile d’estimer l’étendue des modifications à réaliser pour modifier un composant.
- La structure du programme est très rigide, l’évolution du programme est difficile et coûteuse.

La solution au couplage fort

- Pour éviter un couplage fort, on préconise un couplage faible, ce qui se traduit par l'utilisation de mise en place de standards de communication entre les composants.
- Les échanges doivent imposer le moins de contraintes possibles aux composants impliqués.

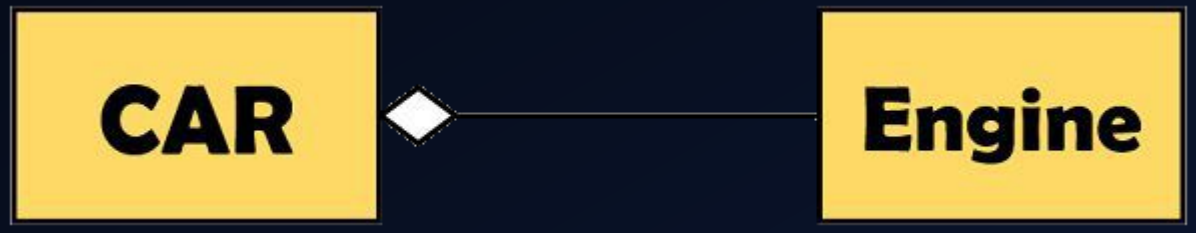
Quelques notions liées à la Poo

AGRÉGATION ET COMPOSITION

L’agrégation

- C’est une sorte de relation entre deux classes où une classe implémente des instances de l’autre.
- Une classe est en relation d’agrégation avec une autre lorsqu’elle a un attribut qui en est une instance ou une collection d’instances.
- Dans une relation d’agrégation, l’enfant peut vivre sans le parent.

Ex: Une classe « PromotionCnam » possède des instances de la classe « Étudiant », mais à la destruction de la classe « PromotionCnam », les instances de la classe « Étudiant » continuent d’exister.



La composition

- C’est une sorte de relation entre deux classes où une classe implémente des instances de l’autre, comme pour l’agrégation.
- Toutefois, c’est une relation plus forte que l’agrégation, car les éléments reliés entre eux n’existent pas l’un sans l’autre.

Ex: Une classe « Bâtiment » possède des instance de la classe « Pièce », et les pièces ne peuvent exister en dehors du bâtiment.

Agrégation vs. Composition

- Pour l’agrégation, l’enfant peut exister indépendamment du parent, contrairement à la composition.
- Le type de relation de l’agrégation est « possède… », le type de la composition est « fait partie de… »
- Dans le cas d’une agrégation, l’enfant peut avoir plusieurs parents, pas dans le cas d’une composition.
- L’agrégation est une relation faible, alors que la composition est une relation forte car elle a plus de contraintes.

Comment les utiliser pour les tests

- Ces relations permettent de limiter le couplage.
- En utilisant l'agrégation, les feuilles peuvent être utilisées dans plusieurs arbres.
- Dans le cas d'une composition, si la feuille ne peut pas être atteinte sans passer par l'arbre, on peut tester par transitivité.
- Dans le cadre du test, une plus grande contrainte peut limiter les tests à faire, car on limite les différents contextes autour du system under test.

Programmer en C#

LES CLASSES ABSTRAITES

Les classes abstraites en C#

- Certaines classes mères existent uniquement pour factoriser des caractéristiques entre d'autres classes filles. Elles ne peuvent pas être instanciées.
- On nomme ces classes des classes abstraites.
- Une classe peut être déclarée abstraite grâce au mot clé « abstract ». Elle peut contenir des fonctions abstraites, c'est-à-dire non implémentées.
- Les classes dérivées, qui sont concrètes, elles, implémentent ces méthodes qui deviennent des méthodes concrètes.

Les classes abstraites

- Exemple de classe abstraite:

```
abstract class Animal
{
    0 références
    Animal(int age)
    {
        Age = age;
    }
    1 référence
    public int Age { get; private set; }
}
```

Programmer en C#

LES INTERFACES

Les interfaces en C#

- Elles sont définies par le mot clé « Interface »
- Elles ne représentent pas un type mais un « contrat »
- Elles peuvent contenir des des fonctions et des méthodes seulement
- Elles peuvent implémenter un nombre illimité d'interfaces
- Elles peuvent être implémentées par autant d'interfaces qu'on veut

Les interfaces en C#

- Elles sont TOUJOURS PUBLIQUES
- Par convention elles commencent toujours par un « I » majuscule.
- Elles ne contiennent que la définition des méthodes, il faut coder l'implémentation dans les classes qui implémentent les interfaces.

Les interfaces en C#

Exemple d’implémentation simple

- Interface

```
1  interface IAnimal
2  {
3      string Name { get; }
4      void Move();
5  }
```

- Implémentation

```
1  class Dog : IAnimal
2  {
3      private string m_name;
4      // On implémente la propriété Name accessible en lecture.
5      public string Name
6      {
7          get { return m_name; }
8      }
9
10     public Dog(string name)
11     {
12         m_name = name;
13     }
14
15     // On implémente la méthode Move.
16     public void Move()
17     {
18         Console.WriteLine("{0} bouge.", m_name);
19     }
20 }
```

Les interfaces en C#

Exemple d’implémentation multiple

- Interface

```
interface IControl
{
    void Paint();
}
interface ISurface
{
    void Paint();
}
```

- Implémentation

```
class SampleClass : IControl, ISurface
{
    void IControl.Paint()
    {
        // Implémentation du Paint appelé par un objet IControl
    }
    void ISurface.Paint()
    {
        // Implémentation du Paint appelé par un objet ISurface
    }
}
```

```
SampleClass obj = new SampleClass();

// La conversion se fait automatiquement.
IControl c = obj;
// Appelle la méthode IControl.Paint de SampleClass.
c.Paint();

// Là-aussi, la conversion se fait automatiquement.
ISurface s = obj;
// Appelle la méthode ISurface.Paint de SampleClass.
s.Paint();
```