# POO généralités.

**POO GÉNÉRALITÉS.....1**

STRUCTURE D'UNE CLASSE.....1
CONSTRUCTEUR.....1
L'OBJET.....2
CRÉATION ET DÉFINITION D'UNE CLASSE.....3
SPÉCIFICATION ET IMPLÉMENTATION.....4
DISCUSSION SUR LE POSITIONNEMENT D'UN SERVICE.....4

Ce document n'est pas un polycopié, c'est seulement un résumé des principales notions abordées en cours, il vient en complément du cours d'amphi et ne contient pas toutes les notions et les illustrations qui ont été présentées.

## *Structure d'une classe.*

Lorsqu'une déclaration est précédée du mot clef static, cette information concerne la classe, tous les services définis dans la classe auront accès à cette information. On parle alors de services ou de variables de classes (par opposition aux services ou variables d'instances).

Static : Contexte de classe

Rien : Contexte d'objet ou d'instance.

```
Public class Rationnel {
        private int numerateur;  // variable d'instance spécifique à un objet particulier
        private int denominateur;  // variable d'instance spécifique à un objet particulier
        public void reduction () {  // service d'instance spécifique à un objet particulier
        .....
        }
```

## *Constructeur.*

Syntaxiquement le constructeur est un service qui porte le même nom que la classe et qui ne possède pas de type de retour. Le rôle du constructeur est d'initialiser les valeurs des variables d'instances d'un objet. On dit aussi que le constructeur donne l'état initial d'un objet.

***Remarque:*** *Pour construire un objet (instancier), on peut avoir besoin d'information en provenance de l'extérieur. Pour cela, il est nécessaire de pouvoir disposer de plusieurs constructeurs différents, on parle alors de surcharge des constructeurs.*

Exemple:

```
public class Rationnel{
        private int numerateur;
        private int denominateur;
        public Rationnel() // constructeur sans paramètre.
        {
```

```
  numerateur = denominateur = 1; // on crée l'élément neutre.
 }
public Rationnel(int num, int deno) // un autre constructeur qui surcharge le précédent.
{
  numerateur = nu;
  denominateur = deno;
 }
```

Une classe définit un nouveau type qui peut être instancier (créer des objets à partir d'une classe). Pour créer un objet, il faut utiliser le mot clef **new**. Par exemple,

Rationnel r = new Rationnel();

Rationnel  r1 = new Rationnel(2,3);

La création d'un objet se fait en deux temps:

1. Réservation de la place mémoire nécessaire à cet objet (en fonction du type de l'objet).
2. Appel à un constructeur en fonction des paramètres (résolution de la surcharge). Le constructeur est appelé pour initialiser l'objet (lui donner son état initial).

Sur cet exemple, le constructeur sans paramètre est appelé, la variable r référence alors un objet rationnel qui a pour valeur l'élément neutre. La variable r1 référence maintenant un objet rationnel dont la valeur initiale est 2/3.

## *L'objet.*

Un objet est instance d'une unique classe, c'est à dire qu'il est créé par une classe. Comme une classe définit un type, **un objet au moment de sa création aura comme type réel** le type de la classe qui l'a instancié. Par exemple, lorsqu'on écrit new Rationnel(), on vient de créer un nouvel objet instance de la classe Rationnel, **le type réel de cet objet est définitivement Rationnel.**

Un objet a un **comportement**, c'est à dire l'ensemble des services d'instances qui ont été définis par la classe qui a créé le nouvel objet. Pour évoquer le comportement associé au **unService** d'un objet **O** on utilise la syntaxe:

**O.unService(p1,...pn);**

On dit que l'objet O est l**e sélecteur ou aussi le récepteur du service**. On demande à l'objet O de **sélectionner et d'exécuter** le service associée au mot clef **unService** en fonction des paramtres p1,..pN. On peut remarquer que comme l'objet participe activement à la sélection du service, on peut avoir des services de même nom mais dans des classes différentes. Comme le nombre et le type des paramètres interviennent aussi dans la sélection du service, on peut avoir plusieurs fois le même nom de service dans une même classe mais avec des paramètres de type et/ou de nombre différents. On parlera alors de **surcharge.**

Lorsqu'on évoque un service d'instance sur un objet O. Le service s'exécute dans l'environnement de l'objet. Cet objet est présent dans le code du service, il est appelé **this** (ce qui correspond au **self** vu en C). Par contre, pour les services de classe, l'objet **this** n'existe pas, puisqu'il ne fait pas partie de la sélection.

Un objet a aussi un état, il y a deux types d'état:

1. L'état logique. L'état logique d'un objet dépend de son état initial (celui donné par le constructeur au moment de l'instanciaton de l'objet) et de la succession des services qu'il a rendu. Les services appliqués sur un objet font donc évoluer son état.
2. L'état physique. C'est la valeur des variables d'instances d'un objet.

Par exemple, si on considère la création d'un objet Rationnel par r = new Rationnel(4,8). L'état physique est représenté par le couple (4,8), alors que l'état logique consiste à dire que r est dans l'état ½. Si on fait une réduction de r par le service r.reduire(). L'état physique change, car maintenant il est représenté par le couple (1,2) mais son état logique n'a pas changé.

## *Création et définition d'une classe.*

Une classe permet de définir un nouveau type, d'instancier des objets, d'appeler les services sur cet objet. Elle se fait en plusieurs étapes de réflexion.

- **Définition de l'interface fonctionnelle publique.** Il s'agit de trouver l'ensemble des services que pourra rendre la classe. Cette interface fonctionnelle est constituée des services publics de classe et des services publics d'instances. Il n'est pas toujours facile de distinguer entre un service publique de classe ou un service publique d'instance. Prenons par exemple, la division de deux rationnels.
    - Service de classe: Rationnel res = Rationnel.division(p,q); On doit interpréter cet appel, comme demander à la classe Rationnel de rendre le service division en considérant deux rationnels qui sont p et q. Le service de classe division rend alors un nouvel objet de type Rationnel qui est le résultat de la division.
    - Service d'instance:Rationnel res = p.division(q); Dans ce cas, on demande à l'objet p de rendre le service associé à la division en prenant comme paramêtre l'objet q, l'objet p a en charge de créer un nouvel objet, de faire le calcul et de rendre le résultat.

  Sur cet exemple, il n'y a aucune raison de demander à un objet de rendre ce service, il est donc préférable d'en faire un service de classe. Une autre différence essentielle, entre service de classe et service d'instance est la suivante:
    - Les services de classes sont décidés à la compilation.
    - Les services d'instances sont vérifiés à la compilation et décidés à l'exécution.
- **Implémentation de l'interface fonctionne publique.** Une fois l'interface fonctionnelle publique de la classe définie, il est nécessaire de l'implémenter. Il faut se rappeller que l'implémentation est un choix temporaire qui peut évoluer au fil du temps ou même changer radicalement. Ce n'est pas le cas, de l'interface fonctionnelle publique qui elle doit respecter la compatibilité ascendante tout au long de son évolution. Les choix d'implémentation portent alors sur :
    - Le choix des variables d'instances ou de classes qui vont permettre de définir l'implémentation.
    - Le choix des services privés (classes/services) qui vont permettre d'assurer une bonne modularité du code (éviter la duplication, permettre la factorisation, accroître

la lisibilité).

- **Ecriture du code des constructeurs:** Une fois, les variables d'instances choisies, il faut maintenant choisir les constructeurs, c'est à dire comment définir l'état d'un objet au moment de sa création et surtout quelles sont les informations nécessaires pour pouvoir créer l'état.

  **Public Rationnel() // création de l'élément neutre.**

  **Public Rationnel(int n) // création d'un rationnel numérateur = n, denominateur =1**

  **Public Rationnel(int n, int d) // création d'un rationnel numérateur = n, denominateur =d**

## *Spécification et implémentation*

Dans tous les cas de figure, l'implémentation d'une classe doit-être conforme au spécification. Par exemple, pour les rationnels est ce que l'on utilise la définition suivante. Un rationnel est un couple d'entiers relatifs, tel que le dénominateur ne soit jamais nul. Dans ce cas, le code du constructeur est alors le suivant :

```
Public Rationnel(int n, int q) {
this.numerateur = n; // on utilise this
this. denominateur = q;
}
```

Par contre si les spécifications pour un rationnel sont les suivantes: Un rationnel est un couple d'entiers relatifs, tel que le dénominateur ne soit jamais nul et tel que le numérateur et le rationnel sont premier entre eux. Dans ce cas, le code du constructeur est alors le suivant :

```
public Rationnel(int n, int q) {
        this.numerateur = n; // on utilise this
        this. denominateur = q;
        this.reduire().
        }
private void reduire(){
        int n = //on calcule le pgcd de numerateur et de denominateur.
        this.numerateur /= n;
        this.denominateur /=n;
}
```

En effet, pour être fidèle aux spécifications, il est nécessaire de réduire, le rationnel au moment de sa création. Mais les spécifications influent sur l'interface fonctionnelle publique d'une classe. Si on reprend l'exemple de Rationnel, dans le second cas, le service d'instance réduire est défini pour éviter la duplication de code, puisque toutes les opérations qui feront évoluer un rationnel devront appeler réduire. Dans le cas 2, ce service ne peut être que privé puisque par définition un rationnel est réduit et il serait stupide de mettre ce service dans l'interface fonctionnelle publique.

Par contre, dans le premier cas, où il n'est pas obligatoire qu'un rationnel soit réduit, il est nécessaire que ce service fasse partie de l'interface fonctionnelle publique.

## *Discussion sur le positionnement d'un service.*

Comme nous l'avons vu précédemment le service réduire à besoin d'un service qui est le calcul du

PGCD de deux nombres entiers relatifs. Plusieurs solutions sont envisageables pour positionner et donner une visibilité au service PGCD.

1. La première solution consiste à écrire directement le code du calcul du PGCD dans le code du service réduire. Cette solution est mauvaise car elle engendre de la duplication de code et nuit à la lisibilité du code.
2. La deuxième solution consiste à en faire un service privé d'instance à l'intérieur de la classe Rationnel.

   ```
   private int pgcd();
   ```

   Dans ce cas, on augmente la lisibilité du code, mais il peut tout de même exister de la duplication de code au sein de l'application, car si on a besoin du calcul du PGCD à l'extérieur de la classe Rationnel, il faudra écrire le code.
3. La troisième solution consiste à en faire un service public d'instance à l'intérieur de la classe Rationnel.

   ```
   public  int pgcd();
   ```

   On diminue maintenant la duplication de code, mais on augmente le couplage, puisque pour calculer le pgcd des entiers p,q, nous sommes obligés de créer un nouveau rationnel:

   ```
   Rationnel tmp = new Rationnel(p,q);
   int pgcd = tmp.PGCD();
   ```

4. La quatrième solution est d'en faire un service public de classe de la classe Rationnel

   ```
   static public int pgcd(Rationnel r);
   static public int pgcd(int a, int b);
   ```

   Le premier prototype n'est pas plus intéressant que la solution 3, car il est toujours nécessaire de créer un objet de type Rationnel, pour obtenir le calcul du PGCD. Par contre, le deuxième prototype qui lui considère un couple d'entier est plus efficace car il ne nécessite pas la création d'un nouveau rationnel, pour calculer le PGCD de deux entiers. Par contre, il est difficile de comprendre pourquoi le calcul du pgcd serait un service public de la classe Rationnel, il n'y a pas de réelles justifications, simplement le fait que le service réduire à besoin du calcul du PGCD, autant le prototype est le bon autant le positionnement de ce service dans la classe Rationnel est problèmatique.

5. La dernière solution consiste à déplacer le service de calcul du PGCD dans une classe bibliothèque.

   ```
   Public class ArithmetiqueLib {
       static public int PGCD(int a, int b).....
       .....
   }
   ```

   Cette dernière solution est la bonne, car le service est correctement positionné et il ne nécessite aucun couplage pour son utilisation.