INFO0201

Introduction à la programmation orientée objet

Partie 2

Membres de classe

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Equipe pédagogique : J. Jonquet, Ch. Jaillet,, P. Delisle, C. Rabat, C. Besançon

Plan

- Membres de classe
  - Attributs et méthodes
  - Diagramme de classes
  - Représentation mémoire
  - Retour sur les exemples
  - Modificateurs (synthèse)

Attribut de classe

Objectif

Définir un attribut dont la valeur sera partagée par toutes les instances de cette classe

=> Attribut commun à tous les objets

Ex : Pour une classe Cercle, la valeur PI

Un compteur d'instance de la classe

Le min et le max d'un ensemble d'objets d'une même classe

Attribut de classe

- Un seul exemplaire en mémoire, avec une valeur dès sa déclaration
- Indépendant de tout objet de la classe
- Mot clé : ***static***

Attribut d'instance

- Autant d'exemplaires en mémoire que d'instance de la classe
- Dépend de l'objet instancié
- Mot clé : *aucun*

Un attribut de classe doit exister complètement avant qu'on crée des instances, donc on doit lui donner une valeur lors de sa déclaration.

Attribut de classe

Utilisation

- Depuis une méthode de sa classe : comme un attribut d'instance
- Depuis une autre classe (classe de test, …) :
  Si **privé** => inaccessible
  si **public** => notation pointée
  - À partir du nom de la classe

```
double pi = Cercle.PI;
```

Un attribut de classe peut être manipulé sans qu'un objet de cette classe ne soit créé

  - A partir d'une instance de la classe

```
Cercle c = new Cercle(…)
double d = c.PI;
```

Un instance dispose des "éléments collectifs" de sa classe

Méthode de classe

Objectif

Méthode pouvant être appelée indépendamment de toute instance de la classe, voire sans instance de la classe

- Indépendant de tout objet de la classe
- Mot clé : ***static***
- Peut accéder qu'à des attributs ou méthodes ***static***

  => ne peut pas accéder à des membres d'instance

Ex : Pour une classe Cercle, une méthode getPI()

```
class Cercle {
        private static double PI = 3,14;
        …
        public static double getPI() { return PI; }
…
}
```

Méthode de classe

Utilisation comme les attributs de classe

- Depuis une méthode de sa classe : comme une méthode d'instance
- Depuis une autre classe (classe de test, …) :
  Si **privé** => inaccessible
  si **public** => notation pointée
  - À partir du nom de la classe

```
double x = Math.sqrt(5.5);
```

Un méthode de classe peut être appelée sans qu'un objet de cette classe ne soit créé

  - A partir d'une instance de la classe

```
Cercle c = new Cercle(…)
double d = c.getPI();
```

Un instance dispose des "éléments collectifs" de sa classe

Exemple

Une classe Truc

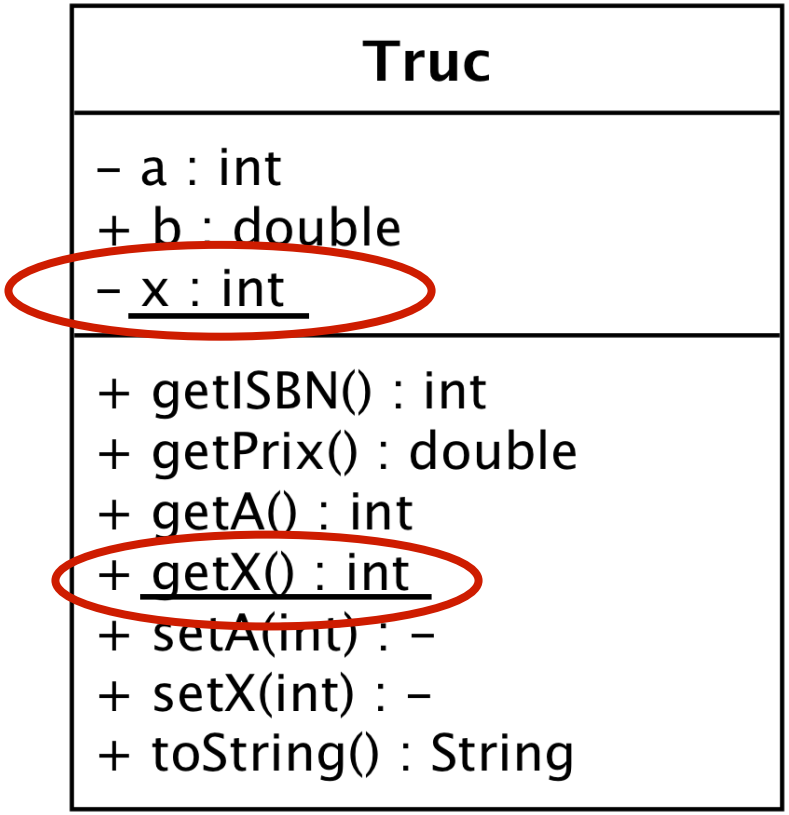
- Deux attributs d'instance : 1 privé et 1 public => a et b : un entier, un réel
  Un attribut de classe : privé => x : un entier (valeur 4)
- Méthodes (d'instance / de classe) : getter(s) + setter(s) + toString

```
class Truc {
    // déclarations des variables
    private int a;
    public double b;
    private static int x = 4;

    //constructeur par init.
    public Truc(int a, double b){
            this.a = a;
            this.b = b;
    }
    //getters
    public int getA(){ return a; }
    public static int getX(){ return x; }
    //setters
    public void setA(int a){ this.a = a; }
    public static void setX(int x){ this.x = x; }

    //affichage
    public String toString(){
            return "a : "+ a + "; b : "+ b +";
x : " + x;
    }
}
```

Diagramme de classe

Les membres de classes (attributs et méthodes) sont soulignés dans le diagramme de classe.

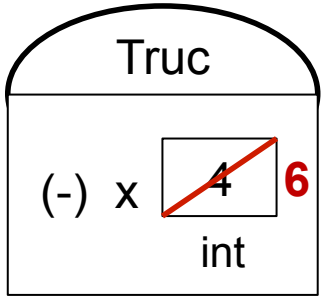
Exemple :



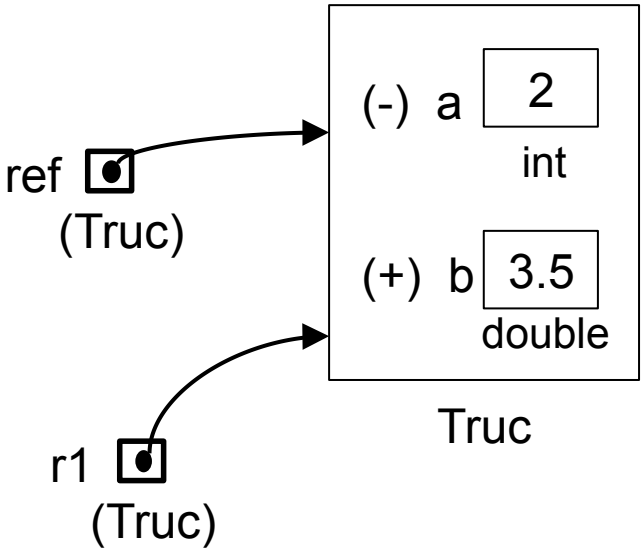
Attributs / méthodes de classe

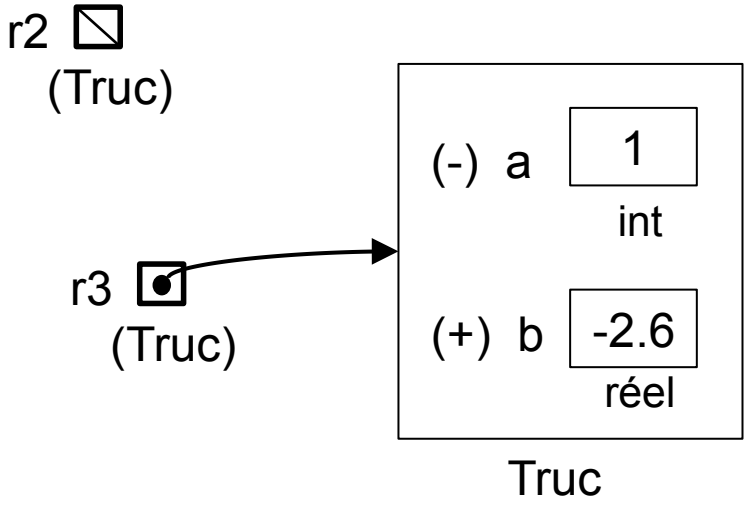
Représentation mémoire

```
\\dans une classe de test
Truc ref = new Truc(2, 3.5);
Truc r1 = ref;
r1.setX(6);
Truc r2;
Truc r3 = new Truc(1, -2.6);
```



1 case mémoire pour toutes les instances de la classe





Retour exemple : les personnes

On souhaite compter le nombre total de personnes générées.

⇒ Ajout d'un attribut de classe nbTotalPers

Peut-on y accéder ? Le modifier ?

⇒ Privé mais une méthode getNbTotalPers() publique

Initialisation ? Quand faut-il l'augmenter ?

⇒ Initialisation à zéro lors de sa déclaration et on l'augmente de 1 dans les constructeurs

```
class Personne {
    //ajout attribut de classe
    private static int nbTotalPers = 0;
    …
    //modification cons. par init.
    public Personne(String n, String p,
                    int a, String ad){
        nbTotalPers++;
        …
    }
    //cons. par defaut appelle le cons. par
    init. => pas de modif

    //ajout getter (methode de classe)
        public static int getNbTotalPers(){
                return nbTotalPers;
        }
        …
}
```

Retour exemple : carte bancaire

On souhaite identifier nos cartes bancaires avec un numéro unique propre à notre classe. Comment faire ?

⇒ Ajout d'un attribut d'instance `identifiant` et d'un attribut de classe `compteur`

⇒ Init. `compteur` : 0 Init. `identifiant` : `compteur`

A chaque nouvelle carte, on augmente le compteur de 1 puis on l'affecte à l'identifiant donc modification des constructeurs et getter pour l'identifiant

```
class CB {
    //ajout attributs
    private static int compteur = 0;
    private int identifiant;
    …
    //ajout getter (methode d'instance)
    public int getIdentifiant(){
        return identifiant;
    }
```

```
    //modification cons. par init.
    public CB(String n, int nu, int c,
                                double p){
        compteur++;
        identifiant = compteur;
        …
    }
//autres cons. Appelle le cons. par init
=> pas de modif
…}
```

Pas de paramètres supplémentaire

Modificateurs : synthèse

- Portée
  - ***public*** => accessible depuis n'importe quelle classe
  - ***Private*** => accessible uniquement depuis la classe où ce membre (attribut/méthode) est défini

- Contexte
  - ***static*** => *de classe*
  - non-static => d'instance

- Modification
  - ***final*** => non modifiable
  - non-final => la valeur peut-être modifiée par une instruction

NB : existe aussi pour les méthodes, et même les classes (*cf* prochains cours)

Prochaine partie

Interface et implémentation