



Programmation orientée objet

Les objets

- L'encapsulation
 - Modularité
 - Sécurité
- Héritage
 - Réutilisation du code
- Généricité
 - Traitement *abstrait* de donnée

L'encapsulation

- Concevoir une classe
 - Se demander ce qui est accessible au monde
 - Ce qui doit être caché (implémentation)
 - Définir le moyen de communiquer avec les autres
- Les données membres
 - Peuvent être des types primitifs
 - Ou des objets de n'importe quel classe
 - On parle de **composition** ou d'**agrégation**

L'encapsulation

- L'accès aux données membres doit être contrôlé
- Ecrire des méthodes pour l'accès aux données
- Et pour la modification des données
- Grâce aux modificateur d'accès, on peut empêcher l'accès directe aux données membres en dehors de la classe

Visibilité des données membres

```
Modificateurs class NomDeLaClasse [extends ClasseMere] [implements NomInterface]
{
  Modificateur type nom;
  Modificateur typeRetour nom(typeParam1, ...);
}
```

Modificateur	Signification
<i>par défaut</i>	Visible par toutes les classes du même package
public	Visible par toutes les autres classes
protected	Modificateur de méthode : visible par les classes du même package et les classes héritées
private	Visibles qu'à l'intérieur de la classe
static	Membre partagé par toutes les instances. Méthode qui n'agit que sur les membres static, pas besoin d'avoir un objet pour l'utiliser.
final	Le membre ne peut être définie qu'une fois. La méthode ne peut être redefinie

Conception

- C'est pas facile
- Il n'y pas de recette miracle
- Mais certaine règles pour y arriver

- Motif de conception (*design pattern*)
- Décomposer (diviser pour régner)
- Pensez en terme de modules réutilisables

Surcharge

- Différentes méthodes peuvent avoir le même nom
 - Pour savoir laquelle est appelée, c'est les paramètres qui jouent
 - Donc même nom mais paramètres différents
- Cela permet d'avoir un nommage cohérent
- Attention aux conversions implicite de type

Test de la surcharge

- Créer une nouvelle classe TestSurcharge
 - Chaque méthode écrit simplement sa signature
 - créer une méthode void f1(type param) pour les types
 - char, byte, short, long, float, double
 - créer une méthode void f2(type param)
 - byte, short, long, float, double
 - etc jusqu'à f7 avec seulement double
 - tester pour tous les types :
 - type arg = val; f1(val); f2(val) ...

Pause !

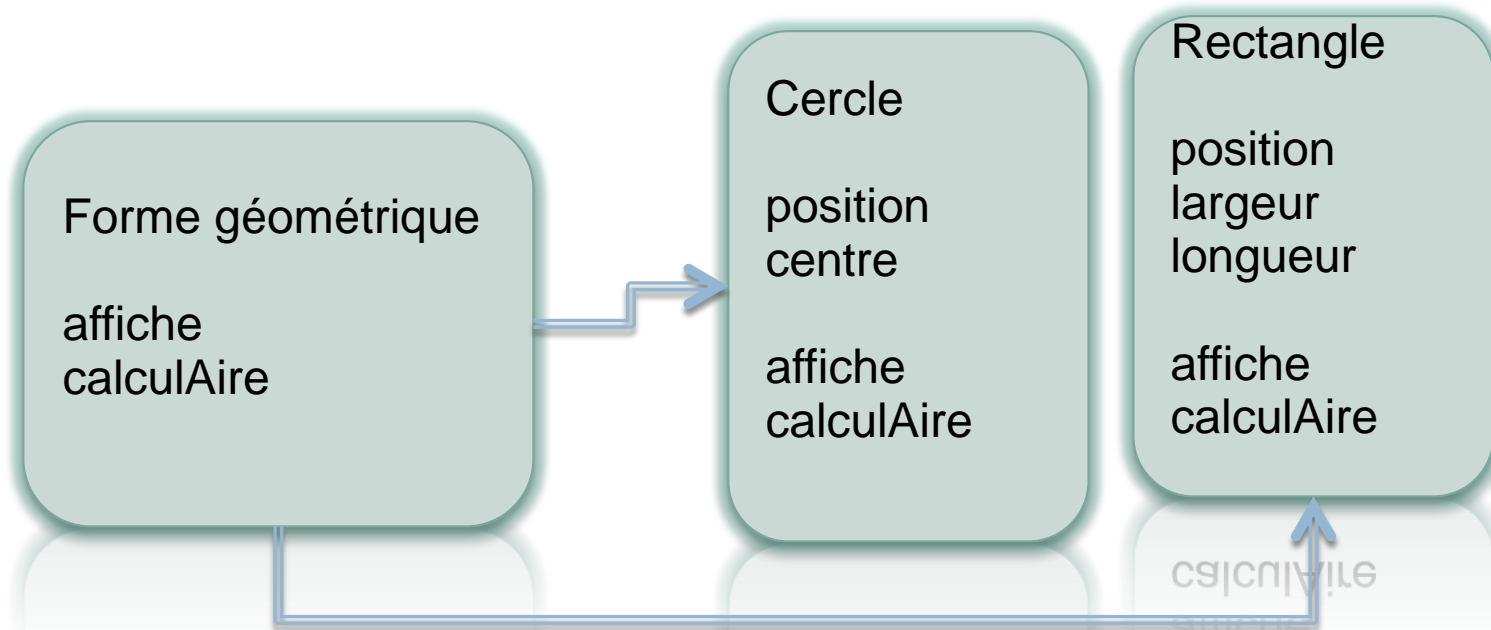


Exercice

- Définir une classe Student
 - Avec les données membres
 - String name; String firstName; String studentNumber
 - Avec une méthode printInformation qui affiche le nom et le prénom
- Tester cette méthode en instanciant un objet Student
- Définir une classe Teacher
 - Données membres ... printInformation
- Définir une classe Car
- Ajouter un membre Car aux classes Student et Teacher
- Définir les modificateur et accesseurs pour les données membres de ces classes

Héritage

- Si une classe est presque comme une autre
 - Réutilisation du code
 - Hiérarchisation du code
- On parle de classe mère et classe fille
 - Les méthodes et données membres sont héritées



Héritage

- Penser l'héritage comme une relation est-un
 - Un rectangle est une forme géométrique
 - Tout ce qui est vrai pour une forme géométrique l'est pour un rectangle
 - Tout ce que l'on peut faire avec une forme géométrique, on peut le faire avec un rectangle
- Une référence d'une classe peut contenir un objet de classe héritée
 - Cela permet le traitement générique des données

Héritage

- Avec une hiérarchie de classes
 - Polymorphisme
 - Une méthode avec le même nom dans la classe mère et dans la fille
 - Un algorithme écrit pour la classe mère fonctionnera pour la classe fille, même si la fille spécialise les méthodes
- Classe abstrait : déclare des méthodes sans les définir

Conception de l'héritage

- Factoriser les éléments commun
- Spécialiser dès que possible
- Lors de l'utilisation de classe
 - Prendre en paramètre la classe de plus bas niveau correspondant à l'algorithme

Exemple : une classe de liste

- Une veut concevoir des listes de ...
 - Une classe mère Liste
 - Des classe filles ListeDe
- Dans la classe mère des méthodes *abstraites*
 - Pour accéder à l'élément suivant
 - Pour savoir si un élément suivant existe
- Les classes filles implémentent ces méthodes

Exemple : une classe de liste

- Pour le traitement des listes
- Parcours de liste générique
- Tri de liste générique
 - Si les élément de la liste sont comparables

En Java

- Visibilité d'une classe

```
Modificateurs class NomDeLaClasse [extends ClasseMere] [implements NomInterface]
{...}
```

```
{...}
```

- Modificateurs :

Modificateur	Signification
abstract	Des méthodes sont abstraites : sans définition. La classe ne peut être instanciée, elle sert seulement de classe mère.
final	La classe ne peut servir de classe mère
private	La classe est locale au fichier ou elle est définie
public	La classe est accessible partout

En Java

- extends : spécifie la classe mère
 - Au plus une classe mère
 - Une classe fille peut surchargé (redéfinir) un méthode de la classe mère (polymorphisme)
- implements : définie des interfaces
 - Une interface est un ensemble de déclaration de méthode
 - Cela permet « l'héritage multiple »

Interface en Java

```
[public/package] interface NomInterface [extends NomInterface, AutreInterface] {  
  // méthode  
  // champs static  
}
```

- Toutes les méthodes sont implicitement abstraites
- Les membres sont implicitement static et final
- Toutes classe qui implémente une interface doit définir les méthodes de l'interface

Exercice

- Définir une classe People
 - Avec les données membres
 - String name; String firstName; Car car; ...
 - Avec une méthode printInformation qui affiche le nom et le prénom, appel printInformation de Car
- Tester cette méthode en instanciant un objet de type People
- Modifier Student et Teacher pour hériter de People

Exercice suite

- Définir une interface `InformationPrint` avec une méthode `printInformation`
 - Modifier les classes définies pour qu'elle implémente la nouvelle interface
 - Créer un tableau de `InformationPrint` et affecter à ce tableau les objets créés.
 - Appeler la méthode `printInformation` pour chaque élément du tableau

Encore du Java

- Les constructeurs
 - new appelle le constructeur de la classe
 - Il permet d'initialiser les données membres
 - De rendre l'objet dans un état cohérent
- Un constructeur est une méthode ayant le même nom que la classe
- Peut avoir des paramètres
- Plusieurs constructeurs possible
 - Appel du constructeur de la class mère : `super()`

On continue l'exo

- Ajouter plusieurs constructeurs aux classes
 - Sans paramètres
 - Avec un ou plusieurs paramètres pour initialiser les données membres



Retour sur les références

Référence et affectation

- Vous vous souvenez des histoires d'adresse ?
- Exemple :

```
Student s1 = new Student();  
Student s2 = new Student();  
s1.name = "a";  
s2.name = "b";  
s2.printInformation();  
s2 = s1;  
s2.printInformation();  
s1.name = "c";  
s2.printInformation();
```

- Il faut faire `s1.name = s2.name`

Clone

- Toutes classe dérive de la classe `Objet`
 - La classe `Objet` contient une méthode `protected clone()`
 - Il faut la redéfinir en `public` pour rendre un objet clonable
 - Et il faut aussi implémenter l'interface `Cloneable`
 - Sinon on a une `Exception`, mais on en parlera plus tard
- La méthode `clone` doit appeler `super.clone()`

Clone

- Enfin il faut cloner les membres objet
 - Soit par l'appel de clone
 - Soit par new et en initialisation les membres

```
public Object clone() {  
    Object o = null;  
    try {  
        o = super.clone();  
    }  
    catch(CloneNotSupportedException e) {  
        e.printStackTrace(System.err);  
    }  
    return o;  
}
```

Encore un exo

- Implémenter la méthode clone pour les classes People, Student et Teacher
- Tester que tout fonctionne bien
 - Instancier deux objet d'une classe
 - cloner l'une dans l'autre
 - modifier les valeurs des données membres

Pour la culture

- Comment fonctionne l'héritage *dans la machine* (une possibilité)
 - Chaque objet représente une adresse
 - A cette adresse on trouve
 - l'adresse du début des méthodes
 - le nombre de méthodes
 - les données membres
- Le compilateur appelle les méthodes par leur adresse, c'est donc dynamique à l'exécution