Institut Polytechnique DEFITECH

SUPPORT DE COURS

INITIATION A LA PROGRAMMATION ORIENTEE OBJET

Niveau : Première année Licence Informatique

Option : Tronc Commun

Dispensé par : AMEVOR Kossi A.

Année académique 2023-2024

Chapitre 3 :

Héritage

I - HERITAGE

➢Définition

Le deuxième grand principe de la programmation objet après **l'encapsulation** est le concept **d'héritage**. Une classe B qui hérite d'une classe A hérite des attributs et des méthodes de la classe A sans avoir à les redéfinir. B est une sous-classe de A ; ou encore A est la super-classe de B ou la classe de base. On dit encore que B est une classe dérivée de la classe A, ou que la classe B étend la classe A. Une classe ne peut avoir qu'une seule super-classe ; il n'y a pas d'héritage multiple en Java. Par contre, elle peut avoir plusieurs sous-classes.

I - HERITAGE

➢Représentation

I - HERITAGE

➢ **Code JAVA**

```
public class A{
}
public class B extends A{
}
```

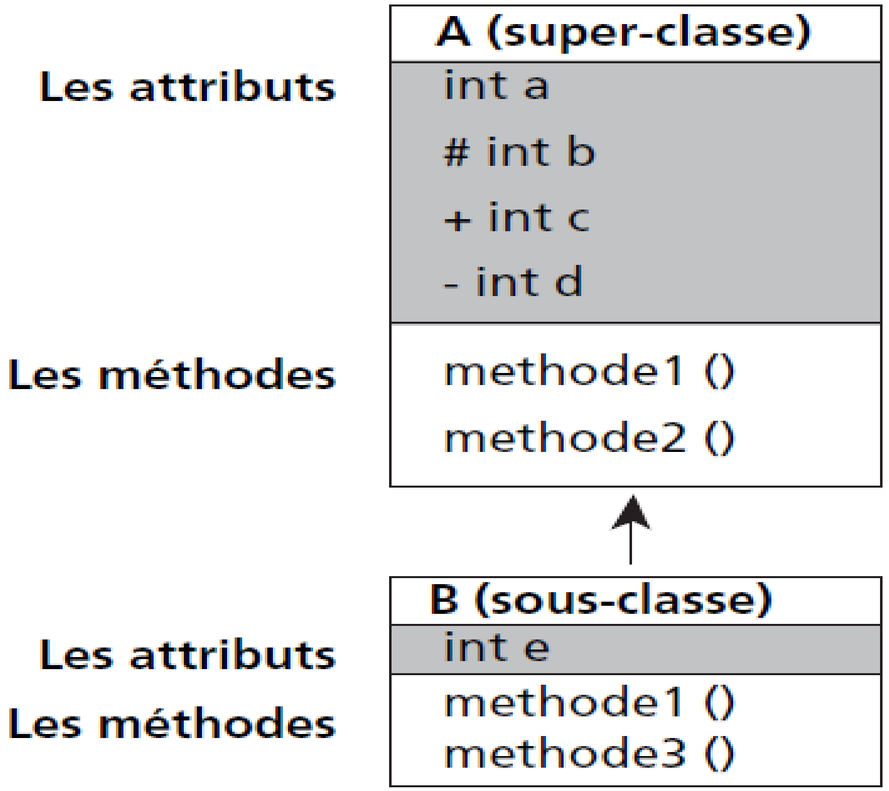
I - HERITAGE

➢Mot clé protected

Le concept d'héritage ajoute un nouveau droit d'accès dit protégé (**protected**) : celui pour une classe dérivée d'accéder aux attributs ou aux méthodes de sa super-classe qui ont été déclarés protected. Un attribut déclaré protected est accessible dans son paquetage et dans ses classes dérivées. Si un attribut de la super-classe est privé (private), la sous-classe ne peut y accéder.

I - HERITAGE

➢Redéfinition de méthode



I - HERITAGE

➢Redéfinition de méthode

- Un objet de la classe B peut accéder aux attributs a, b, c et e (a, b et c sont obtenus par héritage ;
- d privé est inaccessible de la classe B. La classe B contient les méthodes methode1() et methode3() de sa classe B, mais peut aussi accéder aux méthodes methode1() et methode2() de la classe A.
- Soit b1 un objet de la classe B obtenu par : B b1 = new B() :

I - HERITAGE

➢Redéfinition de méthode

Accès aux attributs :

b1 peut accéder :

- à l'attribut c (public : b1.c),
- à l'attribut b (protected dont accessible dans la sous-classe : b1.b),
- à l'attribut a si classe B et classe A sont dans le même paquetage (b1.a).
- b1 ne peut pas accéder à l'attribut d qui est privé (b1.d interdit).

I - HERITAGE

➢Redéfinition de méthode

Accès aux méthodes :

- b1 peut activer une méthode spécifique de B, methode3() par exemple, comme suit : b1.methode3().
- b1 peut activer une méthode non redéfinie de sa super-classe A comme si elle faisait partie de sa classe B. Exemple : b1.methode2().
- methode1() est redéfinie dans la sous-classe B. On dit qu'il y a **redéfinition de méthodes**.
- Contrairement aux méthodes surchargées d'une classe qui doivent se différencier par les paramètres, une méthode redéfinie doit avoir un **prototype identique** à la fonction de sa super-classe. b1 exécutera methode1() de sa classe B sur l'instruction qui suit : b1.methode1().
- L'objet peut aussi activer la méthode de sa super-classe A en préfixant l'appel de la méthode de **super** : **b1.super.methode1().**
- On suppose sur cet exemple que methode1() de la classe A et methode1() de la classe B ont le même nombre de paramètres, chacun des paramètres ayant le même type, et la valeur de retour est la même.

I - HERITAGE

➢ Appel du constructeur de la super-classe

La classe B peut appeler le constructeur de sa super-classe A par le mot clé super()

```
public A(int a, int b, int c, int d) {
    this.a = a;
    this.b = b;
    this.c = c;
    this.d = d;
}
```

I – HERITAGE

➢ Appel du constructeur de la super-classe

```
public B(int a, int b, int c, int d, int e){
    super(a, b, c, d);
    this.e = e;
}
```

II – POLYMORPHISME

➢ Définition

- Le polymorphisme est la faculté attribuée à un objet d'être une instance de plusieurs classes.
- Il a une seule classe réelle qui est celle dont le constructeur a été appelé en premier (c'est-à-dire la classe figurant après le new) mais il peut aussi être déclaré avec une classe supérieure à sa classe réelle.
- Cette propriété est très utile pour la création d'ensembles regroupant des objets de classes différentes.

II – POLYMORPHISME

➢ Exemple

- Créer une classe « Animal » qui possède une méthode ***crier()*** qui affiche simplement « *cri non défini* ».
- Créer ensuite les classes « Coq », « Chien » et « Chat » qui héritent de la classe « Animal » et qui redéfinissent clairement la méthode **crier()** selon le cri de ces animaux.
- Créer un ensemble d'instances de ces animaux et afficher leur cri.

III – CLASSES ABSTRAITES

➢ Définition

Lorsque l'on exploite les possibilités de l'héritage et du polymorphisme, on peut être amené à créer une classe simplement destinée à servir de classe de base pour d'autres classes, et en aucun cas à donner naissance à des objets. Bon nombre de langages permettent alors de « formaliser » cela dans la notion de classe abstraite.

III – CLASSES ABSTRAITES

➢ Définition

Nous conviendrons qu'une telle classe se déclare ainsi :

```
public abstract class nomDeLaClasse{

}
```

IV – CLASSES ABSTRAITES

➢Méthodes abstraites

Pour l'instant, nous avons considéré qu'une classe abstraite disposait classiquement d'attributs et de méthodes, comme n'importe quelle classe. Mais on peut généralement y trouver ce que l'on nomme des méthodes retardées (ou encore abstraites ou différées). Il s'agit en fait de méthodes dont on ne fournit que la signature et le type du résultat. N'étant plus définies dans la classe abstraite, elles devront obligatoirement être définies dans toute classe dérivée pour que cette dernière permette d'instancier des objets.

IV – CLASSES ABSTRAITES

➢ Intérêt

Le recours aux classes abstraites et aux méthodes abstraites facilite largement la conception des logiciels. En effet, on peut placer dans une classe abstraite toutes les fonctionnalités dont on souhaite disposer pour toutes ses descendantes :

- soit sous forme d'une implémentation complète de méthodes (non abstraites) et d'attributs (privés ou non) lorsqu'ils sont communs à toutes ses descendantes,
- soit sous forme d'interfaces (signature + type du résultat) de méthodes abstraites dont on est alors certain qu'elles existeront dans toute classe dérivée instanciable.

IV – INTERFACE

➢ Définition

Dans la réalisation d'une classe, on pouvait distinguer théoriquement son « interface » de son « implémentation ». Rappelons que l'interface correspond à l'ensemble des signatures des méthodes. Bon nombre de langages vont permettre de « formaliser » cette notion d'interface en offrant :

- un mécanisme de définition d'interfaces,
- un mécanisme obligeant une classe à implémenter une interface donnée.
- une même classe pourra implémenter plusieurs interfaces,
- les interfaces pourront jouer un rôle voisin de celui des classes abstraites et participer au polymorphisme (on parlera de « polymorphisme d'interfaces »).

IV – INTERFACE

➢ Définition

```
public interface nomDeL'interface {

}
```

IV – INTERFACE

➢ Implémentation d'une interface

Soit A une classe et I une interface. A implémente l'interface I s'écrit :

```
Public interface I {

}

public class A implements I{

}
```

IV – INTERFACE

➢ Remarques

On notera bien que le fait qu'une classe implémente une interface donnée ne l'empêche nullement de disposer d'autres méthodes. Autrement dit, A n'est pas nécessairement une implémentation de I. On sait simplement que A dispose au minimum des méthodes prévues dans l'interface I.
D'ailleurs, une même classe peut implémenter plusieurs interfaces, comme dans cet exemple :

```
public class A implements I1, I2{

}
```