Héritage

Module : POO
Enseignante: BOUZAROURA Ahlam
Année universitaire: 2019-2020

Introduction

Le deuxième grand principe de la POO après l'encapsulation est le concept d'héritage. Des classes ayant des fonctionnalités communes peuvent être construites en utilisant le mécanisme d'héritage.

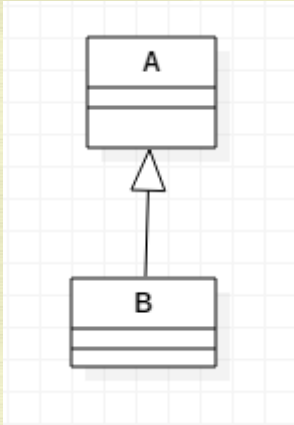
Une classe B qui hérite d'une classe A hérite des attributs et des méthodes de la classe A sans avoir à les redéfinir.

- B est une s***ous-classe*** ou classe ***dérivée*** de A ;
- A est la ***super-classe***(ou ***sur-classe)*** de B ou la ***classe de base*** ou la ***classe mère.***

Héritage

On dit :

- Une **classe B** est dérivée d'une **classe A**
- **B** est une 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.

Avantage de l'héritage

- **La réutilisation du code :**Le code commun entre différentes classes est factorisé dans la classe mère, les modifications se trouvent facilitées.
- **La relation qui relie une classe à sa super-classe:** Une classe B qui hérite de la classe A peut être vue comme Un sous-type (sous ensemble) du type défini par la classe A.

Exemple :

- Un **EtudiantSportif** est un **Etudiant**.
- L'ensemble des étudiants sportifs est inclus dans l'ensemble des étudiants.

Héritage en Java

L'héritage est spécifié lors de la déclaration de la classe `a l'aide du mot clé ***extends***

```
public class UneClasseFille extends UneClasseMere{
...
}
```

Exemple:

```
public class EtudiantsSportif extends Etudiant{
...
}
```

- En Java, toutes les classes sont dérivée de la classe **Object**.

Conséquences d'héritage

- La classe fille contient toutes les méthodes et tous les attributs de la classe mère (même s'il ne sont **pas visibles**).
- Les instances de la classe fille B (objets) sont aussi des instances de la classe mère A.
- La classe fille peut en plus contenir d'autres méthodes ou d'autres attributs ou **redéfinir** ceux de la classe mère.

Droits d'accès et l'héritage

- Un attribut ou une méthode déclaré **protected** est accessible dans son paquetage et dans ses classes dérivées.
- Si un attribut ou une méthode de la super-classe est privé (**private**), la sous-classe ne peut y accéder.

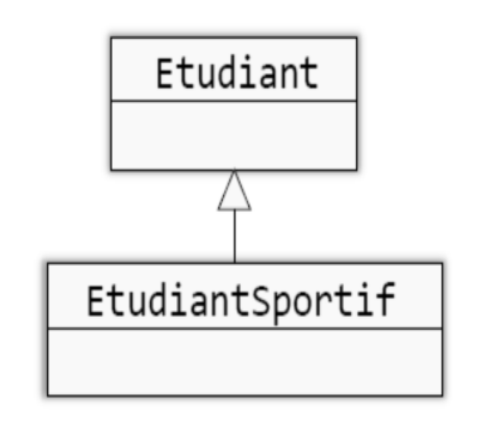
Héritage & Surclassement

Si on a une classe B dérivée de la classe A alors:

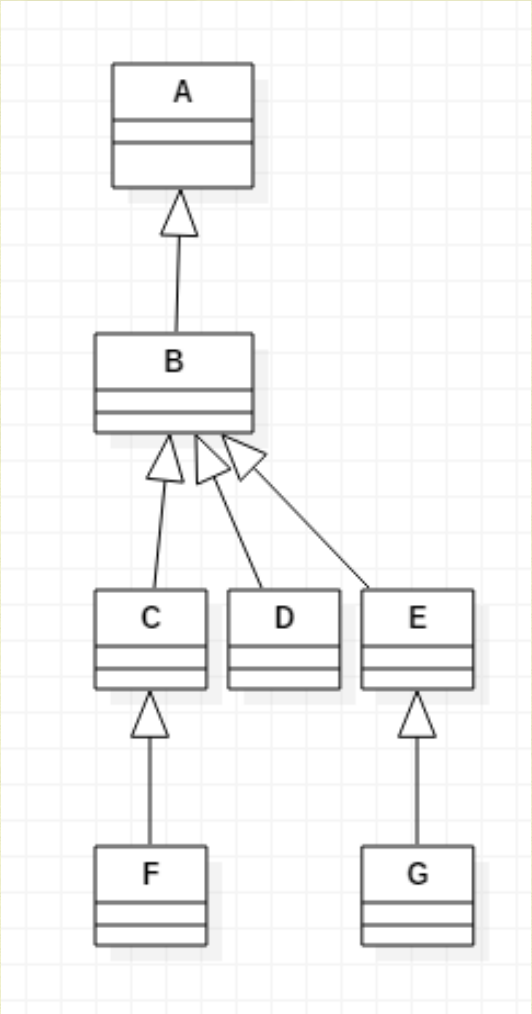
- ✓ tout objet instance de la classe B peut être aussi vu comme une instance de la classe A. Cette relation est directement supportée par le langage JAVA

on a que *EtudiantSportif* est une classe dérivée de la classe *Etudiant* alors en Java on peut écrire:

```
Etudiant e;
e= new EtudiantSportif(…);
```



- **Exemple:** soit la hiérarchie des classes suivantes, On a:




```
A a; B b; C c;
a =new B();☑
a =new F(); ☑
b= new A(); ☒
b= new F(); ☑
c=new E(); ☒
c=new F(); ☑
c=new A(); ☒
c=new B(); ☒
```

Opérateur *instanceof*

L'opérateur ***instanceof*** permet de savoir à quelle classe appartient une instance, c'est un opérateur de comparison il retourne : **true** ou **false**

```
class B{ …}

class D extends B{…}

class C {…}
B b = new B();
D d = new D();
C c = new C();
```

```
b instanceof B // true
b instanceof D // false
d instanceof B // true
d instanceof D // true
```

Redéfinition des méthodes

A l'intérieur d'une classe fille, une méthode implémentée dans la classe mère peut être récrite : c'est la redéfinition.

Dans ce cas, elle remplace la méthode de la classe mère pour les objet appartenant à la classe fille.

La méthode redéfinie doit avoir le même nom, les mêmes attributs (type et ordre) et le même type de retour que ceux de la méthode de la classe mère.

Sans redéfinition de la méthode m	*Avec redéfinition de la méthode m*

class A {
void m(){
System.out.println(
"méthode m de A");
{
}
class B extends A{
 	
}
	class A {
	void m(){
	System.out.println(
	"méthode m de A");
	{
	}
	class B extends A{
	void m(){
	System.out.println(
	"méthode m de B");
	}
	}

A a =new A(); B b=new B(); a.m(); **a = b;** a.m();	A a =new A(); B b=new B(); a.m(); **a = b;** a.m();

Sans redéfinition de la méthode m	*Avec redéfinition de la méthode m*

class A {
void m(){
System.out.println(
"méthode m de A");
{
}
class B extends A{
}
}
	class A {
	void m(){
	System.out.println(
	"méthode m de A");
	{
	}
	class B extends A{
	void m(){
	System.out.println(
	"méthode m de B");
	}
	}

A a =new A();
B b=new B();
a.m();
a = b;
a.m();
	A a =new A();
	B b=new B();
	a.m();
	a = b;
	a.m();

méthode m de A **méthode m de A**	**méthode m de A** **méthode m de B**

Réutilisation du code existant

Dans une méthode redéfinie, il est possible d'appeler la méthode de la classe mère avec le mot clef **super()** dans la première ligne de l'implémentation.

Cette approche est très souvent utilisée dans les **constructeurs redéfinis** ; on profite ainsi de toutes les initialisations faites par la classe mère. Il remplace exactement le nom de la méthode et peut donc être utilisé avec des arguments d'appel.

Classe mère	Classe fille

public class Mere{
void Affiche(String str){
System. out . println ("La mere affiche "+str);
{
{
	public class Fille extends Mere{
	void Affiche (String str){
	super(str);
	System. out . println ("La fille affiche "+ str);
	{
	{

Code	**résultat**
Mere a= new Mere(); Fille b = new Fille (); a. affiche (" Bonjour de A"); b. affiche (" Bonjour de B");	La mere affiche Bonjour de A La mere affiche Bonjour de B La fille affiche Bonjour de B

Classe Object -1-

La classe **Object** est la "mère" de toutes les classes Java. Ainsi lorsqu'on écrit :

```
public class personne
```

on écrit implicitement :

```
public class personne extends Object
```

La hiérarchie d'héritage est un arbre dont la racine est la classe ***Object*** (***java.lang***)

Classe Object -2-

- toute classe autre que **Object** possède une super-classe
- toute classe hérite directement ou indirectement de la classe **Object**
- par défaut une classe qui ne définit pas de clause extends hérite de la classe **Object**
- La classe **Object** n'a pas de variable d'instance ni de variable de classe
- La classe **Object** fournit plusieurs méthodes qui sont héritées par toutes les classes sans exception. Les plus couramment utilisées sont les méthodes **toString** et **equals**

Héritage et constructeurs – 1-

Rappel:

- Les constructeurs sont des méthodes particulières qui portent le même nom que la classe à laquelle elles appartiennent.
- Elles sont automatiquement exécutées lors de la création d'un objet. Le constructeur par défaut ne possèdent pas d'arguments.
- Les constructeurs n'ont pas de type et ne retournent pas.
- Les constructeurs ne sont pas hérités par les classes dérivées.

Héritage et constructeurs – 2-

Chaque instance est munie de deux références particulières :

- **this** réfère l'instance elle-même.
- **super** réfère la partie héritée de l'instance.
- L'appel d'un constructeur de la classe mère doit être la première instruction du constructeur de la classe fille.
- Il n'est pas possible d'utiliser à la fois un autre constructeur de la classe et un constructeur de sa classe mère dans la définition d'un de ses constructeurs.

Héritage et constructeurs –3-

- Si aucun constructeur de la classe ou de la super-classe n'est invoqué explicitement, le compilateur ajoute un appel au constructeur sans argument de la super-classe.

```
public class A {
public A(int x) {
super();
this();
 {
{
```

```
public class B extends A
{
public B(int x) {
//appel super() implicite
this.x = x;
...
}
}
```

chaînage des constructeurs -1-

Lorsqu'un constructeur est exécuté, les constructeurs des classes parentes le sont également. C'est le chaînage des constructeurs:

- si la première instruction d'un constructeur n'est pas un appel explicite à l'un des constructeur de la superclasse, alors JAVA insère implicitement l'appel **super()**
- chaque fois qu'un objet est créé les constructeurs sont invoqués en remontant en séquence de classe en classe dans la hiérarchie jusqu'à la **classe Object**
- Le constructeur de la classe Object qui est toujours exécuté en premier, suivi des constructeurs des différentes classes redescendant dans la hiérarchie

chaînage des constructeurs -2-

Exemple:

```
public class A {
public A () {System.out.println("A");}}
public class B extends A {
public B () {System.out.println("B");}}
```

l'instruction:

```
B b = new B ();
```

produira l'affichage suivant :

```
A
B
```

Interdire l'héritage

Lors de la conception d'une classe, le concepteur peut empêcher que d'autres classes héritent d'elle (classe finale).

```
final public class A { }
```

L'utilisation de mot clè **final** implique que toutes les méthodes de la classe finale sont des méthodes finales (elles ne peuvent être redéfinies)

Remarque:

Les méthodes **private** sont implicitement finales.