جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Chapitre 5

Héritage et Polymorphisme

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

Héritage ?

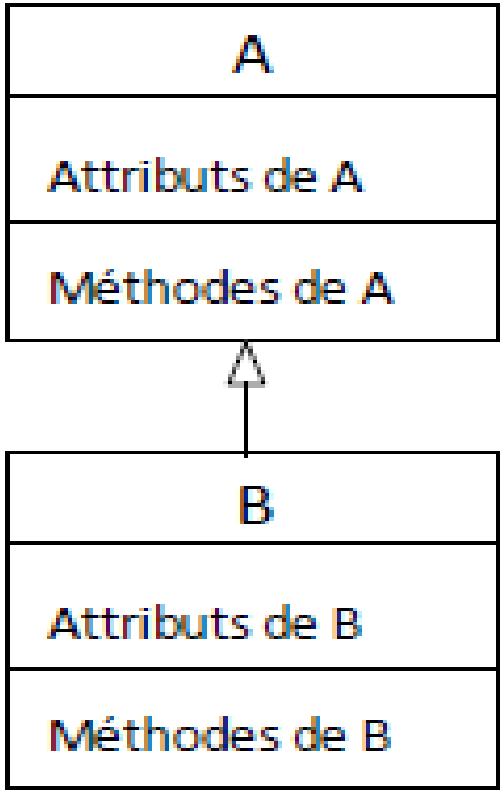
Héritage: principe de l'approche orientée objet (conception et programmation)

- permet de définir des classes par **extension** de classes déjà existantes

- permet la **réutilisation** de code déjà écrit

Héritage

Héritage: exprime le fait qu'une classe B est dérivée à partir d'une classe A



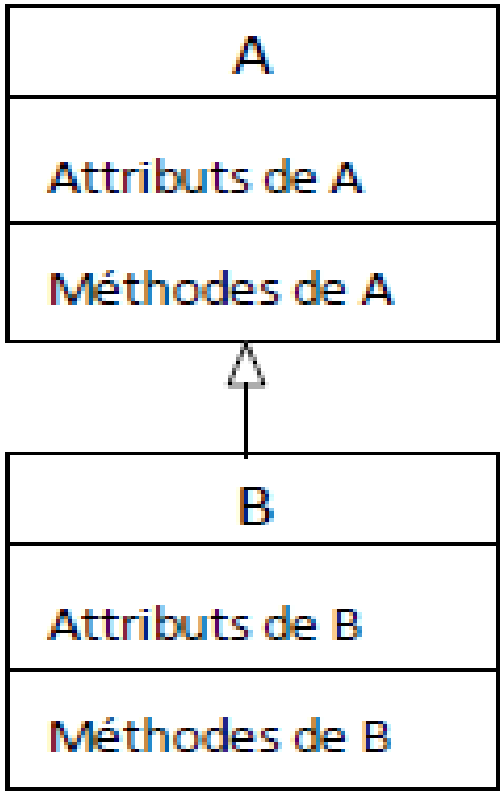
La flèche vers le haut exprime l'héritage

La classe A est une **super-classe** (**classe mère**)

La classe B est une **sous-classe** (**classe fille** ou **classe dérivée**)

Héritage en java

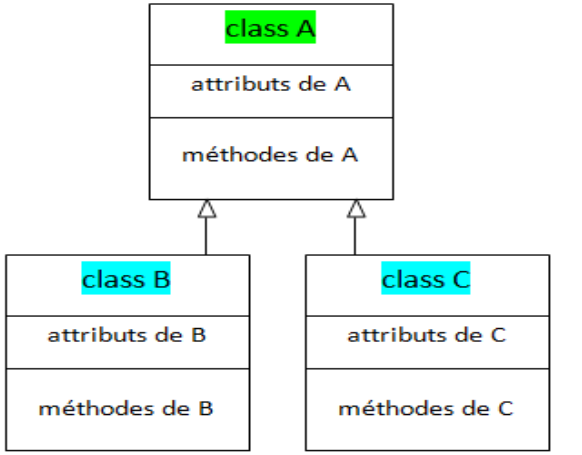
Le mot clé « extends »: exprime l'héritage en java




```
class B extends A
{// attributs spécifiques de B
// constructeur de B
// méthodes spécifiques de B
}
```

Tous les attributs et méthodes **définies dans A sont héritées dans B sans les réécrire**

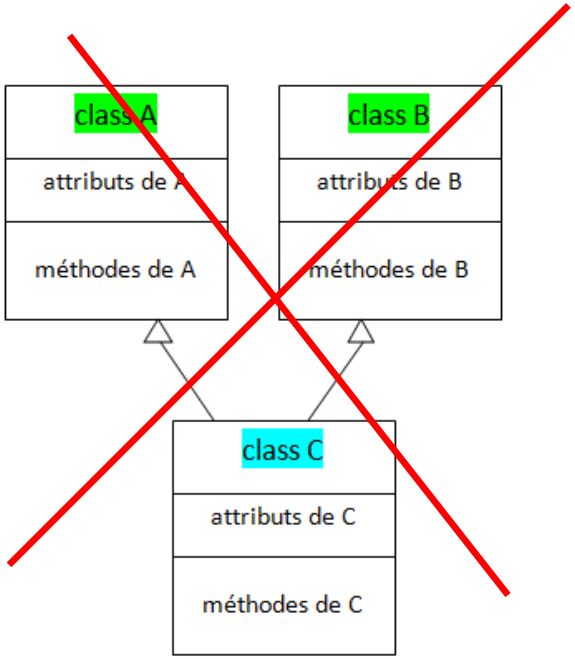
Héritage en java – Règles générales



Une classe (super-classe) peut dérvier plusieurs classes (sous-classes)

class B **extends** A
{// attributs de B
// constructur de B
// méthodes de B
}

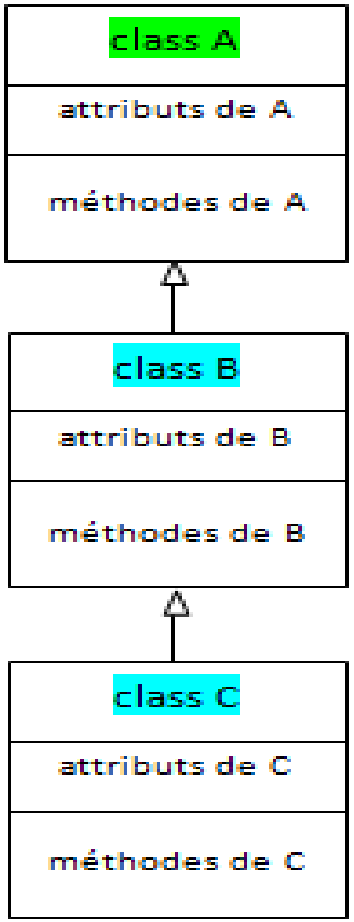
class C **extends** A
{// attributs de C
// constructur de C
// méthodes de C
}



L'héritage multiple n'est pas permis en java

Une classe ne peut être dérivée de deux classes (directement)

Héritage en java – Règles générales



On peut avoir un héritage **multi-niveaux**

Une classe B qui dérive de A

Une classe C qui dérive de B

class B **extends** A

{// attributs de B

// constructur de B

// méthodes de B

}

class C **extends** B

{// attributs de C

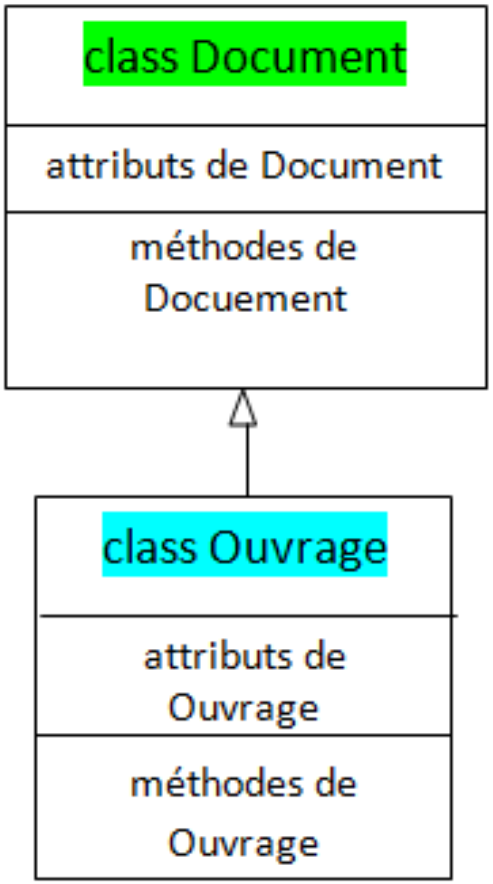
// constructur de C

// méthodes de C

}

La classe C hérite de tous les attributs et méthodes de A et de B

Héritage en java – lien sémantique



Tout objet de la classe B est aussi objet de la classe A

B est un sous-type de A

Relation "is a" ou "est un"

Exemples de liens d'héritage

Un employé **est une** personne

Un ouvrage **est un** document

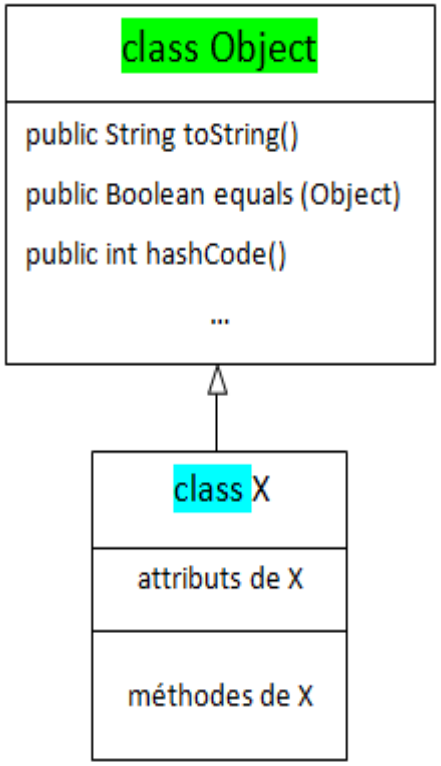
Un cercle coloré **est un** cercle

Un arbre fruitier **est un** arbre

Une Liste FIFO **est une** liste

…

Héritage en java – class Object



Toute classe définie en java **hérite** implicitement de la **superclasse "Object"** définie dans le package java.lang.

On n'a pas besoin d'écrire

Class X extends Object , l'héritage de la classe Object est implicite

Méthodes de la classe Object

public String toString();

public Boolean equals (Object);

public int hashcode ();

…

Héritage en java – Exemple

```
public class Produit
{String ref;
  String libellé;
  float Prix;
// constructeur
// méthodes
}
```

Exemple Produit ("Ref005", "savon liquide", 200.0)

Exemple Produit-Alimentaire ("Ref028", "Lait", 90.00, 03-02-2017, 03-03-2017, "Sec-frais-10°C")

```
public class Produit-Alimentaire extends Produit
{ // attributs supplémentaires
   Date DatFab, DatExp ;
   String Conditions;
// méthodes supplémentaires
}
```

Héritage

Un objet de la classe Produit possède **trois attributs**: réf, libellé, coût.

Un objet de la classe Produit-Alimentaire possède **cinq attributs**: réf, libellé, coût, **dateFab, datePer, conditions**

- Tout objet de la classe Produit-Alimentaire peut accéder à toutes les méthodes définies dans la classe Produit.

Héritage en java – Exemple

```
public class Arbre
{String Nom, Espece, Zone,
CondVie;
// constructeur
// méthodes
String getNom()
{return Nom; }
… }
```

Arbre **A** = ("pin", "…", "méditerranée", "chaleur et humidité")

```
public class ArbreFruitier extends Arbre
{String Nomfruit, Formefruit; Color Cfruit;
// constructeur
// méthodes
String getForme()
      {return Formefruit; }
… }
```

ArbreF**ruitier AF** ("Fraisier", "…", "Est", "Fraise", « triangulaire",Color.red);

Héritage

Un objet de la classe Arbre possède **trois attributs**: Nom, espèce, zone, condVie.

Un objet de la classe ArbreFruitier possède **sept attributs**: Nom, espèce, zone, condVie, nomFruit, forme, cFruit (couleur),

- Tout objet de la classe ArbreFruitier peut accéder à toutes les méthodes définies dans la classe Arbre.

On a défini une méthode spécifique à la classe ArbreFruitier:

String **getForme()**

{return Formefruit; }

Héritage – accès aux attributs/méthodes

- Une méthode définie dans la sous-classe peut faire référence aux attributs définis dans la superclasse (directement ou en utilisant le mot « **super** »)
- De même une méthode d'une sous-classe peut appeler une méthode de la superclasse.

Une méthode définie dans la superclasse ne peut pas utiliser (i.e faire référence) les attributs définis dans la sous-classe

De même une méthode de la superclasse ne peut pas appeler une méthode de la sous-classe

Héritage – Chainage des constructeurs

-**1-** Tout constructeur d'une classe autre que la classe Object fait appel à un **constructeur de sa superclasse**.

- **2-** L'appel au constructeur de la superclasse se fait à l'aide du mot réservé **«super»** considéré comme un appel de méthode.

- **3-** L'appel au constructeur de la superclasse (**super (…)**) doit se trouver en **1ère ligne du constructeur de la sous-classe.**

- **4 - Si aucun appel au constructeur de la superclasse, le compilateur ajoute implicitement l'appel au constructeur par défaut (sans paramètres). Et Si le constructeur par défaut n'est pas disponible, une erreur est générée.**

- **5-** Un constructeur peut aussi faire appel à un autre constructeur de sa classe (surcharge de constructeurs) à l'aide du mot réservé **« this »**

Chainage des constructeurs - Exemple

```
public class Produit
{String ref, libellé;
float Prix;
// constructeur
public Produit ( String ref, String lib, float px)
{ this.ref = ref ; libellé = lib ; prix = px ; }
// méthodes
}
```

```
Produit P = new Produit ("Ref012", "savon liquide", 200.00)
```

```
Produit-Alimentaire PA = new Produit-Alimentaire ("Ref028", "Lait",
90.00, new Date (03, 02, 2019), new Date (03,03,2019) "Sec-frais-10°C")
```

```
public class Produit-Alimentaire extends Produit
{ // attributs supplémentaires Date DatFab, DatExp ; String Conditions;
// constructeur
public Produit-Alimentaire (String ref, String lib, float px, Date DF, Date DE, String cond)
{ super (ref, lib, px) ;        /* appel au constructeur de la superclasse pour initialiser
                                les attributs ref, libellé et prix */
DatFab = DF ; DatExp = DE ; Conditions = cond ;   // initialisation des autres attributs
}
// méthodes
```

Héritage – Accès aux attributs/méthodes

-**1-** Pour accéder aux attributs/méthodes de la superclasse, dans une sous-classe, on utilise le mot clé **«super».**

super.nomattribut
super.nomMethode(…);

- **2-** L'utilisation du mot « super » n'est pas obligatoire, s'il n' y a **pas de redéfinition** d'attributs/méthodes entre sous-classe et superclasse

Héritage – Exemple

```
public class Produit
{String ref;
  String libellé;
  float Prix;
                    // constructeur
public Produit ( String ref, String lib, float px)
        { this.ref = ref ; libellé = lib ; prix = px ; }

                    // méthode Modifier

public void Modifier (String newlib, float newpx)
        { libellé = newlib ; prix = newpx ; }
```

Héritage – Exemple

```
public class Produit-Alimentaire extends Produit
{  Date DatFab, DatExp ; // attributs supplémentaires
   String Conditions;
                                    //constructeur
public Produit-Alimentaire (String ref, String lib, float px, Date DF, Date DE,
String  cond)
{ super (ref, lib, px) ;   /* appel au constructeur de la superclasse pour
initialiser  les attributs ref, libellé et prix */
DatFab = DF ; DatExp = DE ; Conditions = cond ;
                                    // méthode modifier
public void Modifier (String newlib, float newpx, Date newDF, Date newDE)
{ super.Modifier (newlib, newpx);   /* appel de la méthode Modifier de
                                       la classe Produit */
DatFab = newDF ; DatExp = newDE ;}  }
```

Héritage – Appel des méthodes

```
Produit P = new Produit ("Ref012", "savon liquide", 200.00)

P.modifier ("gel moussant", 300.00);
```

// méthode modifier de la classe Produit

```
Produit-Alimentaire PA = new Produit-Alimentaire ("Ref028", "Lait", 90.00,
new Date (03, 02, 2019), new Date (03,03,2019) "Sec-frais-10°C")

PA.modifier ("Lait poudre", 150.00, new Date (10, 05, 2019), new Date
(10,12,2019) "Sec-frais-25°C"
```

// méthode modifier de la classe Produit-Alimentaire

Héritage – visibilité des champs

Question

Si l'on veut attribuer les modificateurs de visibilité **(public, private, protected)** aux différents champs dans les classes Produit et Produit-Alimentaire, lesquels pourrait-on utiliser?

Expliquez.

Héritage – visibilité des champs

Réponse

Si on affecte le modificateur « private » aux différents champs dans la classe Produit, ces champs seront **inaccessibles** dans la classe Produit-Alimentaire

On pourrait utiliser le modificateur **« protected »** pour les attributs de la superclasse afin de donner accès dans les classes dérivées uniquement et qui restent inaccessibles dans d'autres classes

Dans la classe Produit-Alimentaire, on peut appliquer la règle générale (déjà énoncée), les attributs déclarés « private » et les méthodes déclarées « public »

Exercice

1- On demande d'implémenter une classe **Forme** qui décrit les formes géométriques régulières. On suppose qu'une forme possède un centre (de type Point) et une couleur.

2- Implémenter deux classes Carré et Cercle en donnant les spécificités de chacune d'elles. Les traitements à implémenter doivent permettre entre autres, de déplacer, redimensionner (zoomer), calculer le périmètre et la surface de la forme.