Programmation Orientée Objet avec JAVA

Chapitre 5 – Héritage et polymorphisme

Chapitre 5 – Héritage et polymorphisme

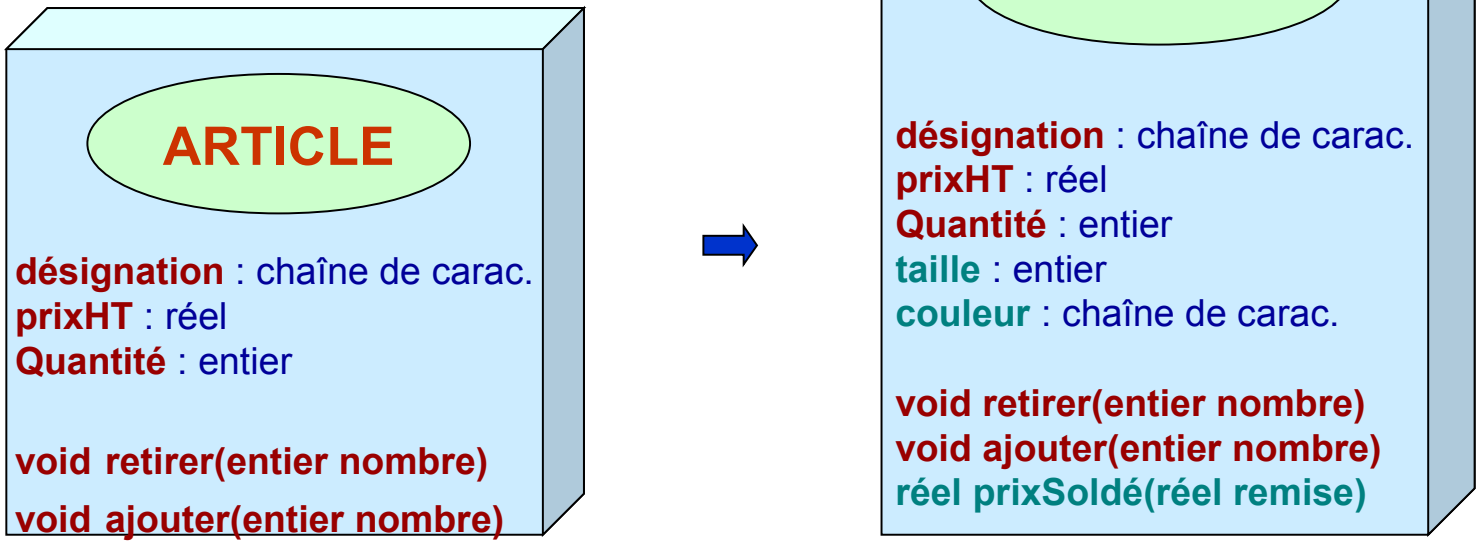
- **Extension du comportement d'une classe (super classe)**
- **Les sous classes héritent de l'état et du comportement de la super classe**
- **Description de la sous classe par extension ou spécialisation de la super classe**

Chapitre 5 – Héritage et polymorphisme

- **Spécialisation par enrichissement**

Réutiliser une classe

- **Exemple de la classe Vêtement**
  - ➥ Spécialisation de la classe Article



Chapitre 5 – Héritage et polymorphisme

- **Spécialisation par enrichissement**
  La duplication

  - **Recopie en adaptant aux besoins**
    - ✓ Substituer les occurrences de la chaîne "Article" par la chaîne "Vêtement "
    - ✓ Ajouter les attributs taille et coloris
    - ✓ Ajouter la méthode prix_soldé(int remise)

  - **Obtention aisée de la nouvelle classe**

  - **Problèmes**
    - ✓ Risques d'incohérences
    - ✓ Maintenance difficile
    - ✓ Surcoût d'archivage

Chapitre 5 – Héritage et polymorphisme

- **Spécialisation par enrichissement**

  L'héritage

  - **La réutilisabilité impose la définition d'un sous-type à partir d'un type existant**
  - **Redéfinition possible pour spécialiser certaines méthodes de la super classe**
  - **Écriture des attributs et des méthodes propres à la sous classe**

Chapitre 5 – Héritage et polymorphisme

- **Comment créer une nouvelle classe à partir d'une classe déjà définie ?**

- **La classe de base peut être personnelle, ou être une classe standard de Java, ou avoir été définie par quelqu'un d'autre.**

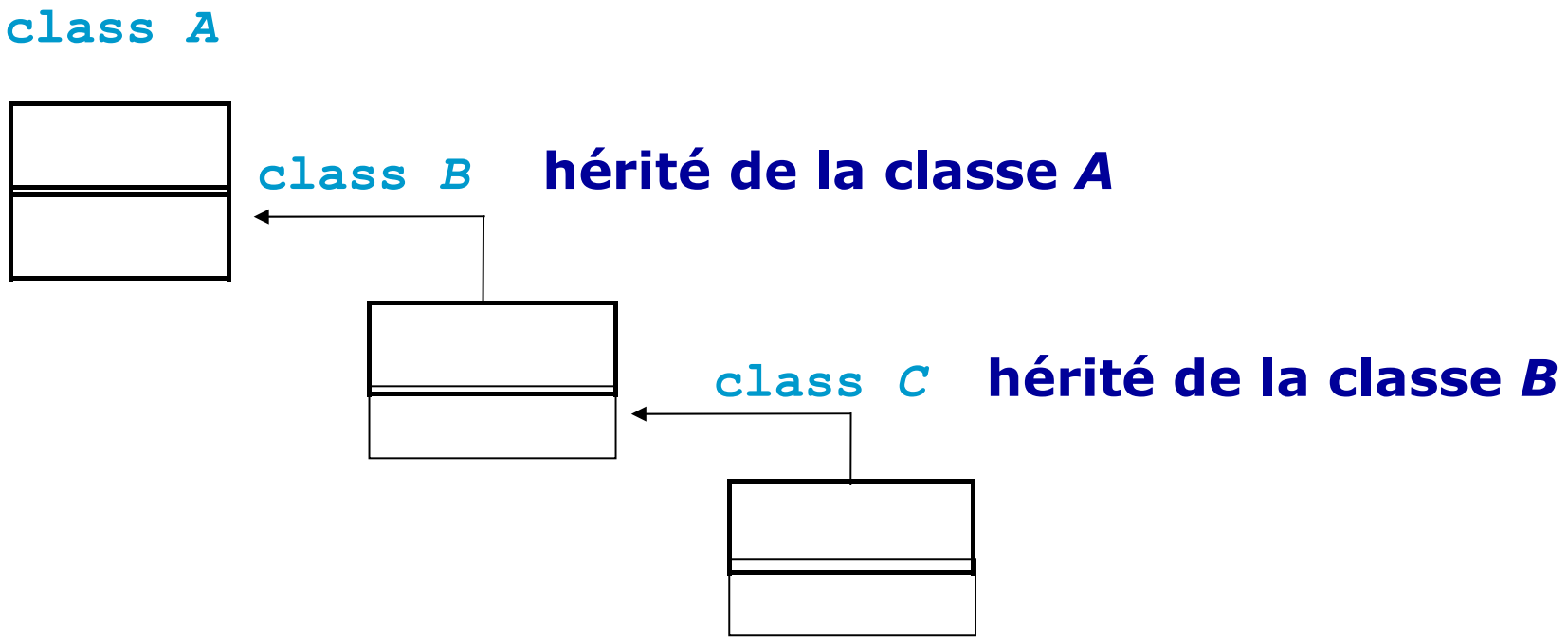
Chapitre 5 – Héritage et polymorphisme

- **Concrètement**
  - **Le mot-clé `extends` permet d'étendre le code d'une classe donnée dite** classe de base**, et de redéfinir certaines méthodes.**
  - **On parle** d'héritage**.**

```
class Derivée extends Base
{
   // nouvelles variables d'objet
   // nouvelles méthodes
   // redéfinitions de méthodes des classes de base
   // constructeurs de type
}
```

Chapitre 5 – Héritage et polymorphisme

- **Héritage d'héritage**



Chapitre 5 – Héritage et polymorphisme

- **Exemple :**

```
    // Fichier Meuble.java
public class Meuble
{
   //définition de la classe Chien
}

    // Fichier Table.java
public class Table extends Meuble
   // définition de la classe Table
   // on définit les propriétés qui les différencient
   // de la classe de base
}
```

Chapitre 5 – Héritage et polymorphisme

- **Héritage simple : une classe dérivée ne peut hériter que d'une classe simple (arbre d'héritage).**
- **Un classe dérivée hérite de l'intégralité du code et des données de la classe de base.**
- **Il faut que ça ait un sens.**
- **Exemples :**

  `Pixel.java`

  `PixeColore.java`

Chapitre 5 – Héritage et polymorphisme

Remarques

- La classe « Mère » de toute les classes Java est « Object »
- Une classe déclarée avec le mot-clé « final » ne peut être dérivée
- La mot-clé « protected » permet d'affiner la visibilité des variables et des méthodes des classes « parent »

Chapitre 5 – Héritage et polymorphisme

- **Exemple**
  - **Représentation des cercles**

```
import java.math;

public class Cercle {
  private double x, y ;
  private double r ;

public Cercle( double x, double y, double r ) {
   this.x = x ; this.y = y ; this.r = r ;
}

 public double perimetre() {
    return 2 * Math.PI * r ;
 }

 public double aire() {
    return Math.PI * r * r ;
 }
}
```

Déclaration des attributs

Déclaration du constructeur

Déclaration des méthodes

Chapitre 5 – Héritage et polymorphisme

- **Exemple**
  - **Utilisation de la classe Cercle dans un programme :**

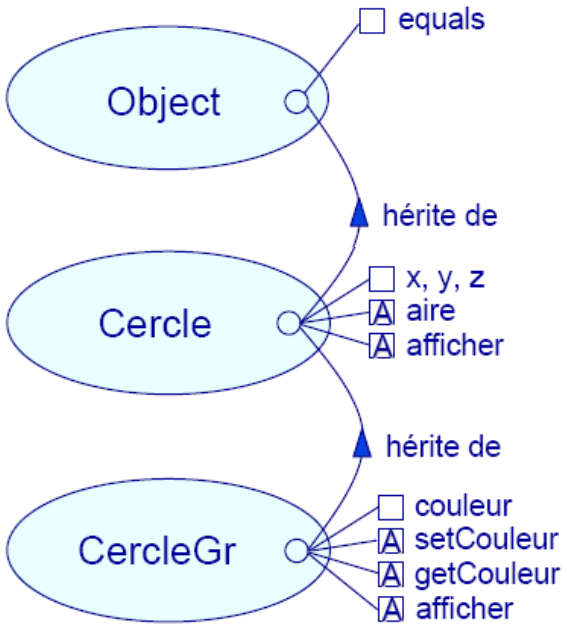
```
public class TestCercle {

public static void main(String[] args) {
   Cercle c  = null;
   c = new Cercle (50,50,100) ;
   double surface = 0.0;
   surface = c.aire ( ) ;
   }
}
```

Chapitre 5 – Héritage et polymorphisme

- **Exemple d'héritage de la classe Cercle**

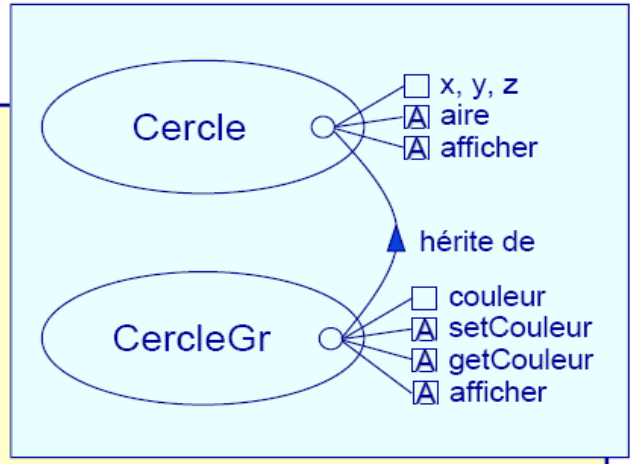
```
public class CercleGr  extends  Cercle   {
    private Color Couleur ;
    public void setCouleur ( ) ;
    public color getCouleur {  return Couleur ;  }

    public void afficher ( ) {
        super.afficher ( ) ;
        …
    }
}
```

Appel du code de la classe « Mère »



Chapitre 5 – Héritage et polymorphisme

- **Utilisation**




```
Cercle c = null ;
CercleGr cg = null ;

cg = new CercleGr ( 23, 67, 234 ) ;
double  p =  cg.aire ( ) ;

cg.setCouleur ( Color.red ) ;
cg.afficher ( ) ;

c = cg ;  // c référence un cercle graphique
c.setCouleur (Color.blue) ;    ((CercleGr) c).setCouleur (Color.blue) ;
perim = c.aire ( ) ;
c.afficher ( ) ;
```

Le code barré est-il juste?

Le down-casting est-il nécessaire?

Chapitre 5 – Héritage et polymorphisme

- **Important :**

  Toute référence sur une classe de base peut référencer un objet d'une sous classe.

- **c = cg; // ok**

- **c.setCouleur(Color.blue); // ok**

Chapitre 5 – Héritage et polymorphisme

Constructeur d'une classe dérivée

- Le constructeur d'une « sous-classe » peut appeler le constructeur de la classe « mère » en utilisant le mot-clé **« super »**
- Le mot-clé « super » doit être à la première ligne du code du constructeur de la « sous-classe »
- Si l'appel a « super » n'est pas mentionné, le constructeur par défaut de la classe « mère » est appelé

Chapitre 5 – Héritage et polymorphisme

Constructeur d'une classe dérivée

```
public class CercleGr  extends  Cercle   {
    private Color Couleur ;

    public CercleGr ( double x, double y, double r, Color Couleur)  {
        super ( x, y, r ) ;
        this.Couleur = couleur ;
    }
}
```

Chapitre 5 – Héritage et polymorphisme

- **Polymorphisme**
  - **Redéfinition dans une classe fille d'une méthode héritée de la classe mère. La signature de la méthode est conservée.**
  - **En Java, accès à la méthode masquée à l'aide du mot clé `super`**
  - **Annotation de redéfinition : @Override**

    Le compilateur Java vérifie que la méthode de la sous-classe a la même signature que celle de la classe mère.

```
@Override
public void afficher(){
    super.afficher();
}
```

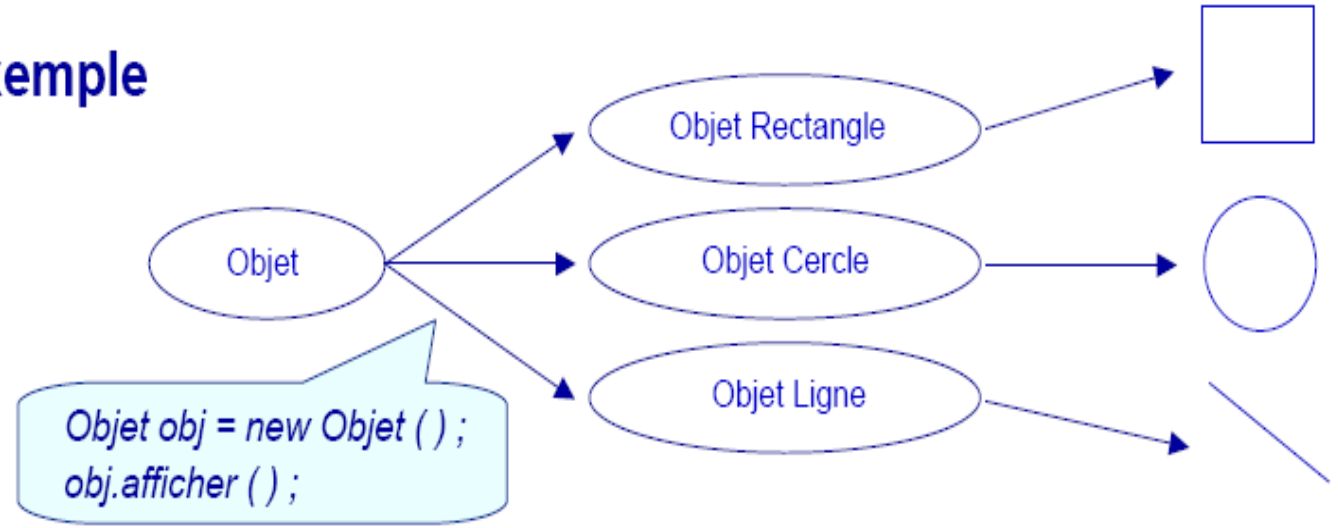
  - **La méthode héritée est masquée par la méthode redéfinie**

Chapitre 5 – Héritage et polymorphisme

Principe

- Un même message envoyé vers divers objets peut déclencher des actions (méthodes) pouvant prendre des formes différentes
- Le choix de la méthode a déclencher est effectué au moment de l'exécution. On parle de **liaison dynamique**

Exemple



Chapitre 5 – Héritage et polymorphisme

- **Les classes abstraites**

Définition

- Une méthode abstraite est une méthode sans code devant absolument être redéfinie dans une des sous-classes
- Une classe abstraite contient au moins une méthode abstraite et ne peut être instanciée
- Définie par le mot-clé « **abstract** »

Chapitre 5 – Héritage et polymorphisme

Déclaration

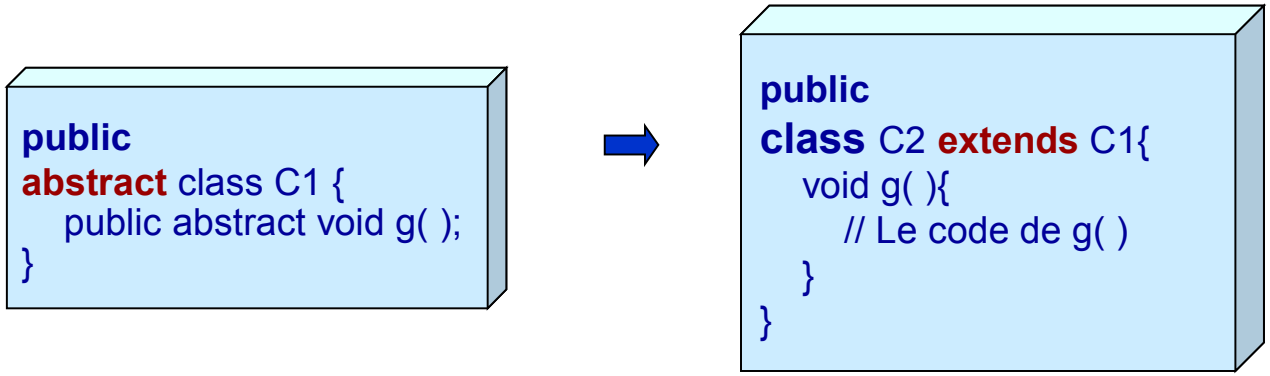
```
public abstract class ObjetGraphique  {
   public Color  couleur ;
   public void  abstract  dessiner ( Graphics g ) ;
   public Cercle ( ) { x = 0.0 ; y = 0.0 ;  r = 0.0 ; }
   public void setCouleur ( Color c ) { couleur = c ; }
   public Color getCouleur ( ) { return couleur ; }
}
```

Remarque : une classe abstraite peut ne posséder aucune méthode abstraite! C'est assez rare!

Chapitre 5 – Héritage et polymorphisme

Utiliser les classes abstraites

- **Pour ne plus être abstraite, une classe fille doit implémenter toutes les méthodes abstraites de la classe dont elle hérite.**



Chapitre 5 – Héritage et polymorphisme

Déclaration

```
public abstract class ObjetGraphique  {
   public Color  couleur ;
   public void  abstract  dessiner ( Graphics g ) ;

   public Cercle ( ) { x = 0.0 ; y = 0.0 ;  r = 0.0 ; }

   public void setCouleur ( Color c ) { couleur = c ; }
   public Color getCouleur ( ) { return couleur ; }
}
```

Chapitre 5 – Héritage et polymorphisme

Utilisation

```
public class CercleGraphique  extends ObjetGraphique  {
      public void  dessiner ( Graphics g )  {
            g.DrawOval ( x, y, r, r ) ;
      }
}
```

```
public class RectangleGraphique  extends ObjetGraphique  {
      public void  dessiner ( Graphics g )  {
            g.DrawRect ( x, y, l, h ) ;
      }
}
```

Chapitre 5 – Héritage et polymorphisme

Utilisation

Instanciation impossible sur une classe abstraite

```
ObjetGraphique  obj ;
obj = new ObjetGraphique ( ) ;

CercleGraphique  c  = new CercleGraphique (…)  ;
RectangleGraphique  r = new RectangleGraphique (…) ;

obj = r ;
obj.dessiner ( g )

obj = c ;
obj.dessiner ( g )
```

Chapitre 5 – Héritage et polymorphisme

- **Modificateurs d'accès**

Modificateurs	Visibilité
public	Une classe, une méthode ou un membre sont visibles et accessibles n'importe où dans le monde Java
Par défaut	Une classe, une méthode ou un membre sont visibles par seules les classes du même package
protected	Une méthode ou un membre d'une classe sont visibles par les classes du même package et par les classe filles se trouvant hors du package
private	Une méthode ou un membre d'une classe sont visibles par cette seule classe

Chapitre 5 – Héritage et polymorphisme

Héritage multiple

- La classe fille hérite de plusieurs classes mères
- Complexe surtout dans le cas du polymorphisme
- Impossible dans certains langages (Java, Smalltalk)

Chapitre 5 – Héritage et polymorphisme

- **Alternative :**
  - **Héritage unique par étapes**

```
class C {
…
}
class B extends A {
…
}
class C extends B {
…
}
```

- **Implémentation des interfaces**

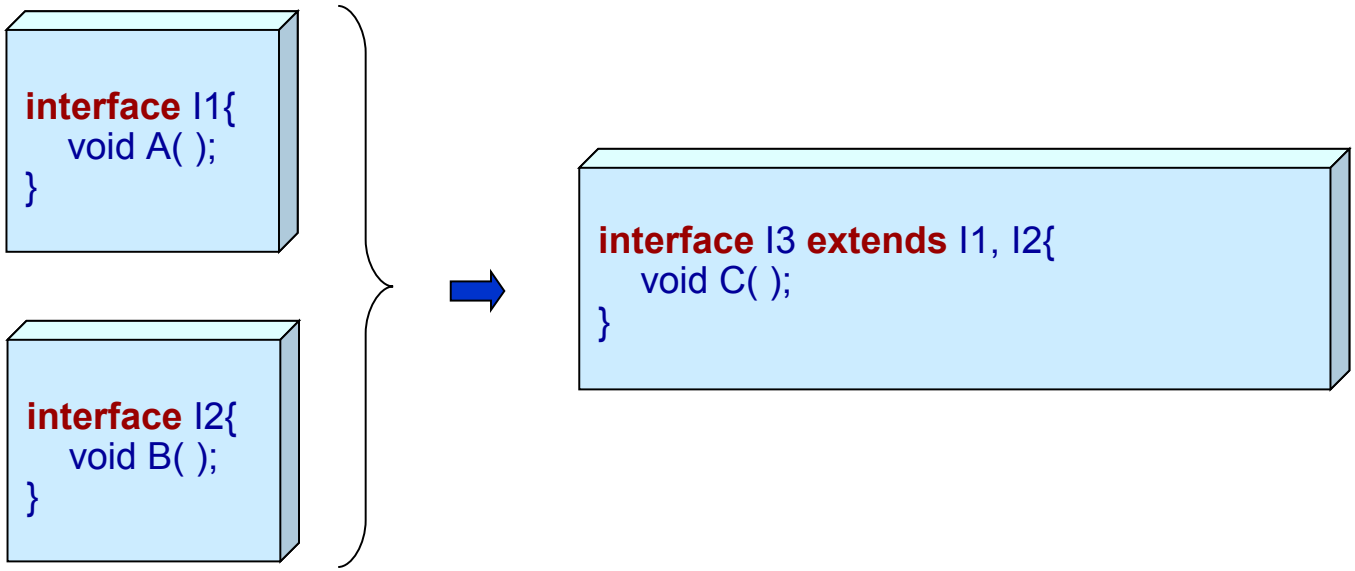
```
interface A {
…
}
interface B {
…
}
class C implements  A, B {
…
}
```

  - **Une classe abstraite permet de définir dans une classe de base des fonctionalités communes à tous ses descendants.**
  - **Si l'on considère une classe abstraite n'implémentant aucune méthode ni aucun attribut (sauf des constantes) alors on parle d'interface.**
  - **Une classe peut implémenter plusieurs interfaces.**

Chapitre 5 – Héritage et polymorphisme

Héritage multiple
utilisation des interfaces java

- **Contrairement aux classes d'instances, une interface peut hériter de plusieurs interfaces**



Chapitre 5 – Héritage et polymorphisme

- **Une classe qui implémente une interface doit implémenter toutes les méthodes de l'interface**