POO-IG

Programmation Orientée Objet et Interfaces Graphiques

Cristina Sirangelo
IRIF, Université Paris Diderot
cristina@irif.fr

Exemples et matériel empruntés :

* Core Java - C.Horstmann - Prentice Hall Ed.
* POO in Java - L.Nigro & C.Nigro - Pitagora Ed.

Héritage

Héritage et réutilisation de code

Héritage : un autre mécanisme de réutilisation de code

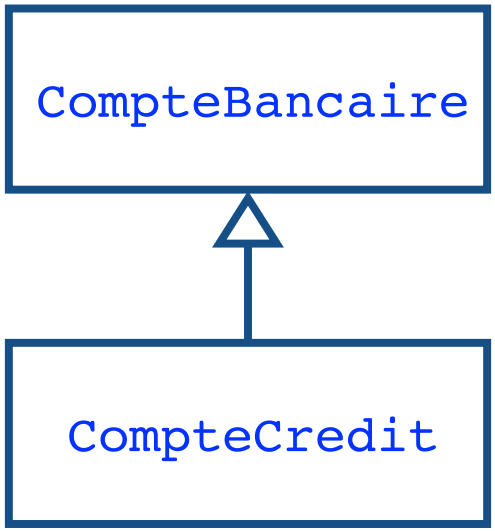
- Certaines classes sont par nature des spécialisation d'autres classes :
  - ont les mêmes propriétés et méthodes + d'autres
- Exemple. Supposons :
  - d'avoir une classe CompteBancaire
  - de vouloir concevoir une nouvelle classe CompteCredit
- Un compte avec crédit est un compte bancaire
  - => la classe CompteCredit aura toutes les propriétés et méthodes de CompteBancaire plus d'autres spécifiques

- L'héritage permet de décrire la classe CompteCredit par différence :
  - décrire uniquement les caractéristiques en plus ou différentes par rapport à l'autre classe

Héritage : syntaxe et notation

```
public class CompteBancaire { … }
public class CompteCredit extends CompteBancaire { …
   // champs et méthodes additionnelles ou différentes
}
```

- CompteCredit : sous-classe (ou classe enfant)
- CompteBancaire : super-classe (ou classe parent)
- Tous les membres de la classe parent sont automatiquement membres de la classe enfant (hérités)
- On dit qu'il existe une relation **is-a** (est-un) entre CompteCredit et CompteBancaire



Exemple de classe parent

```
package poo.banque;

public class CompteBancaire {
  private String numero;
  protected double solde = 0;
  public CompteBancaire(){numero ="";}
  public CompteBancaire( String numero ){
     //pre-condition: numero != null
     this.numero = numero;
   }
  public CompteBancaire( String numero, double solde ){
     //pre: numero != null && solde >= 0
     this.numero = numero; this.solde = solde;
   }
  public void verser( double montant ){
     if( montant<=0 ) throw new IllegalArgumentException
                                   ("montant non valide.");
     solde = solde + montant;
   }
 //continue…
```

Exemple de classe parent

```
// Suite : Class CompteBancaire
public boolean retirer( double montant ){
    if( montant <= 0 )
     throw new IllegalArgumentException("montant non
                                              valide.");
    if( montant > solde ) return false;
    solde -= montant;
    return true;
}
public double getSolde(){
    return solde;
}
public String getNumero(){
    return numero;
}

}//CompteBancaire
```

Conception d'une sous-classe

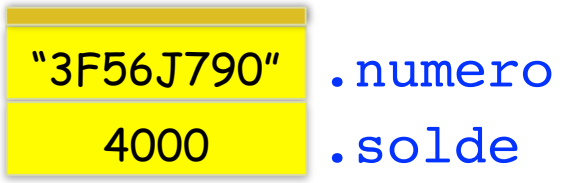
```
package poo.banque;
public class CompteCredit extends CompteBancaire {…}
```

- La classe `CompteCredit` hérite de `CompteBancaire` tous les champs et méthodes (pas les constructeurs)
- Donc par simple `extends` un compte avec credit a un solde un numéro et toutes les méthodes pour les manipuler
- En plus un compte avec credit doit permettre un découvert, jusqu'une limite (crédit)

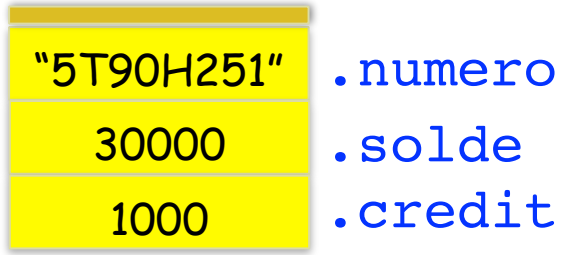
```
public class CompteCredit extends CompteBancaire {
    private double credit = 1000;
    //constructeurs et nouvelles méthodes
    …
}
```

Objets de sous-classe

Un objet de classe `CompteBancaire`

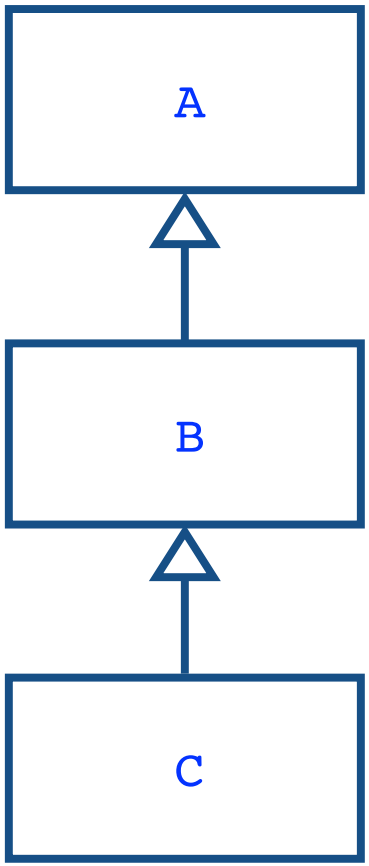


Un objet de classe `CompteCredit`



Hiérarchie de classes

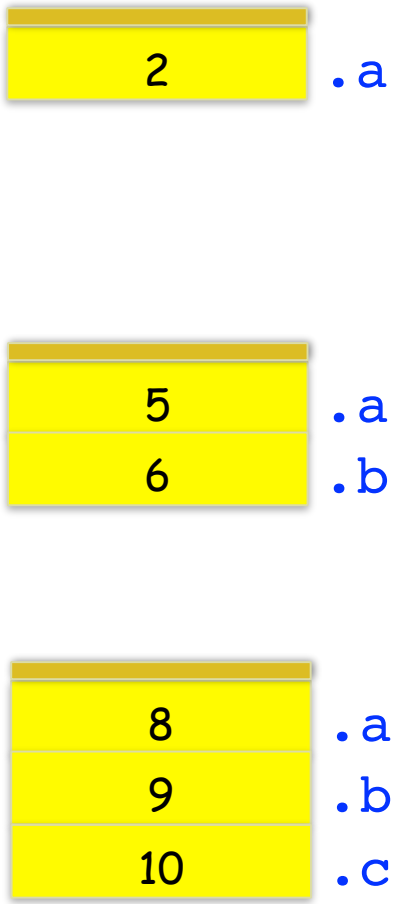
Diagramme



Définition de la classe

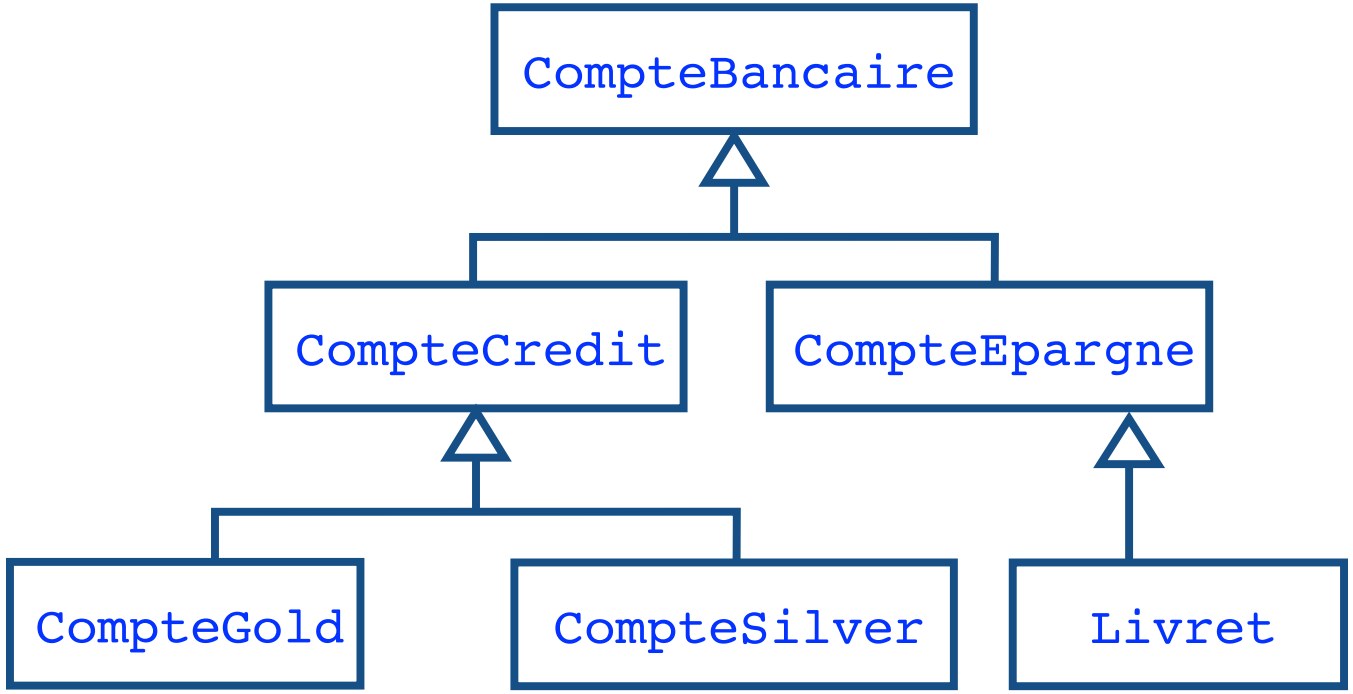
```
class A {
    int a;
    …
}

class B extends A {
    int b;
    …
}

class C extends B {
    int c;
    …
}
```

Exemple d'objet de la classe



Hiérarchie de classes

- En Java une classe peut avoir plusieurs sous-classes
- Mais une classe peut avoir une seule classe parent
- => La hiérarchie est en général arborescente



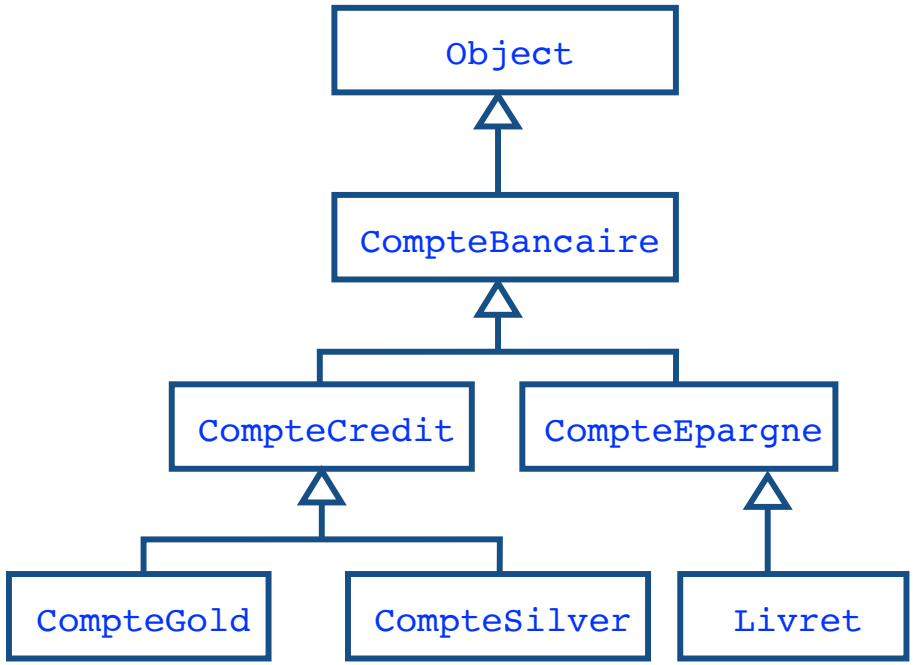
Hiérarchie de classes

- La classe `Object` (dans `java.lang`) est la racine de cette hiérarchie
  - une classe qui n'étend aucune autre classe, étend implicitement Object

```
class CompteBancaire
{…}
```

equivalent à

```
class CompteBancaire
      extends Object
{…}
```



  - => Toute classe étend directement ou indirectement Object

Méthodes additionnelles

- Les méthodes de la classe parent sont héritées par la classe enfant, qui peut en plus définir des méthodes additionnelles

```
public class CompteCredit extends CompteBancaire {
    private double credit = 1000;
    //constructeurs
    …
    public double getCredit() { return credit; }
    public void nouveauCredit( double credit ){
        if( credit < 0 || solde + credit < 0)
            throw new IllegalArgumentException
                                ("Nouveau credit non valide.");
        this.credit = credit;
    }
    //autres méthodes
    …
}
```

Remarque : les méthodes de la classe enfant peuvent manipuler à la fois les champs hérités (visibles) et les champs propres

Méthodes additionnelles

- Les méthodes additionnelles ne sont évidement pas accessibles par un objet de la classe parent

```
CompteBancaire cb = new CompteBancaire (“ASD634”);
CompteCredit cc = new CompteCredit (“SFD78634G”);

cb.getCredit();//ERREUR
cc.getCredit();//OK méthode propre
```

- En revanche les méthodes (visibles) de la classe parent sont également des méthodes de la classe enfant

```
cc.getSolde();// OK méthode héritée
```

Constructeurs de sous-classes

- Initialisation d'un objet de sous-classe
  - à la fois les champs hérités et les champs propres doivent être initialisés
- Pour initialiser les champs hérités il est nécessaire d'invoquer le constructeur de la classe parent : `super (…)`

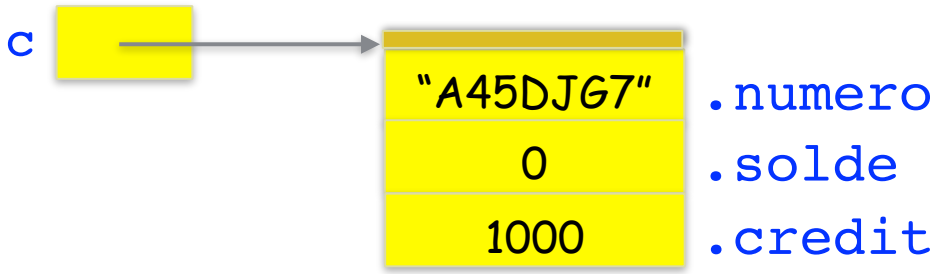
- Si présent, `super(…)` doit être la première instruction du constructeur
- `super()` et `this()` ne peuvent pas être invoqués dans un même constructeur
- Si **aucun des deux** n'est invoqué, un appel `super()` (sans arguments) est implicite au début du constructeur
  - Dans ce cas : erreur si la classe parent n'a pas de constructeur sans arguments

Constructeurs de sous-classes et `super()`

```
public class CompteCredit extends CompteBancaire {
   private double credit = 1000;
   public CompteCredit
                (String numero, double solde, double credit){
      super( numero, solde );
      if( credit < 0 || credit + solde < 0)
                throw new IllegalArgumentException
                              (“credit ou solde  non valide.");
      this.credit = credit;
   }
   public CompteCredit () {} //super() implicite
   public CompteCredit (String numero){
        super( numero );
   }
   public CompteCredit (String numero, double solde){
        super (numero, solde);
   }
   //methodes
   …
```

Créer des objets de sous-classe

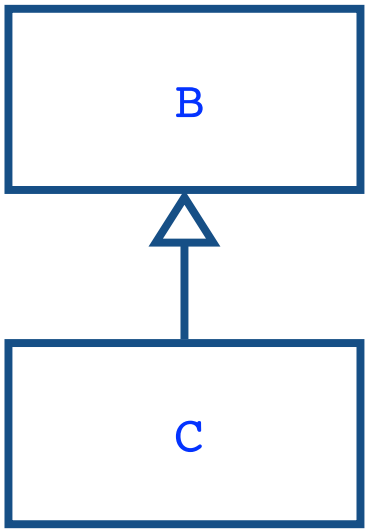
```
CompteCredit c = new CompteCredit ("A45DJG7");
```



Redéfinition de champs

- Un champ avec le même nom d'un champ de la super-classe cache ce dernier

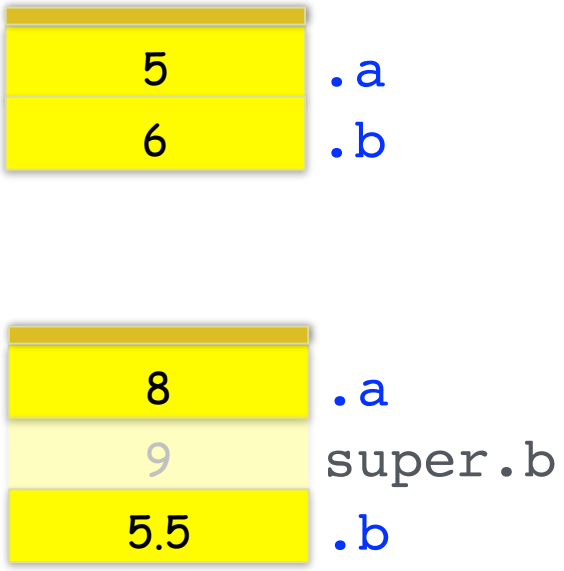
Diagramme



Définition de la classe

```
class B {
    int a;
    int b;
    …
}
class C extends B {
    double b;
    …
}
```

Exemple d'objet de la classe



Redéfinition de champs et `super.`

```
class B {
    int a;
    int b;
    …
}
class C extends B {
    double b;
    void f() {
      b = 5.5;  // champ double de sous-classe
      super.b = 9; // champs int de la super-classe
     }
     …
}

B ob = new B();
C oc = new C();
ob.b = 9; // champs int de la super-classe
oc.b = 5.5; // champ double de la sous-classe
```

Overriding (Redéfinition de méthodes)

- Il est possible de redéfinir aussi des méthodes de la classe parent
- redéfinition : en anglais overriding
- pour qu'il y ait redéfinition, la méthode redéfinie dans la sous-classe doit avoir la même signature (nom de méthode et types des paramètres) que la méthode de la classe parent à redéfinir

```
public class CompteCredit extends CompteBancaire {
    private double credit = 1000;
    //constructeurs, getCredit et nouveauCredit
    …
    public boolean retirer( double montant ){
        //pre: montant > 0;
        if( montant <= solde + credit ) {
            solde -= montant;
            return true;
        }
        return false;
    }
}
```

Overriding et `super.`

- Une méthode redéfinie cache la méthode de la classe parent

```
CompteBancaire cb = new CompteBancaire ("A346",1000);
CompteCredit cc = new CompteCredit ("GA38",1000, 200);

cb.retirer (1200);// retirer de CompteBancaire, retrait refusé
cc.retirer (1200);//retirer de CompteCredit, retrait effectué
```

- Si une méthode (visible) de la classe parent a été redéfinie, elle est encore accessible dans la classe enfant : notation `super.` (e.g. `super.retirer (montant)`)

- Remarque : pas utile dans notre cas (`super.retirer` échoue si le solde devient négatif)

Redéfinition : remarque

- Pour les variables : c'est le nom de la variable qui est pris en compte (pas le type).
  - dans une classe, à chaque nom de variable ne correspond qu'une seule déclaration.
  - même nom dans une sous-classe => redéfinition

- Pour les méthodes : c'est la signature (nom + type des paramètres) qui est prise en compte
  - dans une classe, à une signature correspond une seule définition
  - même signature dans une sous-classe => redéfinition

  - remarque : on peut avoir des méthodes de même nom et de signatures différentes (surcharge)