Séquence 9

Héritage

1 Introduction

L'héritage est un concept-clé de la programmation objet. Il permet de définir une classe en ajoutant des composants supplémentaires à une classe existante. Ces composants sont notamment des attributs et des méthodes. Par ailleurs, il est possible de redéfinir une méthode, c'est-à-dire de conserver le même nom et les mêmes paramètres, mais de changer les instructions de la méthode.

Dans ce chapitre, nous allons commencer l'étude de l'héritage en voyant comment utiliser l'héritage pour définir de nouvelles classes.

2 Présentation de l'exemple des comptes avec découvert

Nous allons réutiliser ici l'exemple des comptes bancaires donné dans le cours sur les classes. Sans redonner tout le code, rappelons les composants de la classe.

— Attributs : numéro (int), titulaire (String) et solde (double).
— Constructeur : `Compte(String tit, int num)`
— Accesseurs : `getSolde, getTitulaire, getNumero`
— Modificateurs : `deposer, retirer`
— Méthode de service : `toString`

Lorsqu'on va écrire une classe qui hérite de la classe `Compte`, nécessairement, il y aura tous les attributs et toutes les méthodes de la classe. Le constructeur n'est pas hérité, il faut en écrire un autre. Certaines méthodes peuvent être redéfinies.

Nous allons définir une classe par héritage depuis `Compte`. On dira que la nouvelle classe est une *sous-classe* ou une *classe fille* de `Compte` et que `Compte` est la *super-classe* ou la *classe mère* de la nouvelle classe.

Nous allons définir une nouvelle classe de compte dans laquelle on vérifiera que le compte ne dépasse pas un découvert fixé, ce découvert pouvant être différent d'un compte à un autre. Pour ce faire, il faut ajouter un attribut `decouvertAutorise`, un accesseur `getDecouvert` et un modificateur `setDecouvert`. Par ailleurs, il faudra redéfinir la méthode `retirer` pour tenir compte du découvert autorisé.

3 Écriture de la sous-classe

Voici le code de la classe.

```
1  public class CompteDecouvert extends Compte{
2      private double decouvertAutorise = 500;
3      public CompteDecouvert(String tit, int num){
4          super(tit,num);
5      }
6      public double getDecouvert(){
7          return decouvertAutorise;
8      }
9      public void setDecouvert(double decouv){
10         if (decouv<0){
11             throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
12         }else{
13             decouvertAutorise = decouv;
14         }
15     }
16     public void retirer(double montant){
17         if (montant<0){
18             throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
19         }else if (solde -montant < -decouvertAutorise){
20             throw new IllegalArgumentException("Dépassement␣du␣découvert␣autorisé");
21         }else{
22             solde = solde -montant;
23         }
24     }
25 }
```

Le fait que l'on utilise lhéritage apparaît sur la première ligne avec la clause `extends Compte`. Cela signifie que la classe `CompteDecouvert` va hériter des attributs et méthodes de `Compte`.

Dans le corps de la classe, l'attribut `decouvertAutorise` ainsi que sont accesseur et son modificateur sont définis. Il y a également un constructeur.

Le constructeur de la sous-classe doit nécessairement appeler le constructeur de la super-classe en utilisant le mot-clé `super`. Ce mot-clé est suivi des paramètres qu'il faut donner au constructeur de la super-classe. Il faut nécessairement que cet appel apparaisse comme première instruction du constructeur de la sous-classe, avant toute autre instruction. Cet appel au constructeur de la super-classe ne correspond pas à un `new` : il ne crée pas un nouvel objet de la classe `Compte`, mais il contribue à créer l'objet instance de `CompteDecouvert`. C'est-à-dire que les instructions du constructeur de la super-classe `Compte` sont exécutées sur un objet instance de `CompteDecouvert`.

La méthode `retirer` est redéfinie pour empêcher de passer au-delà du découvert autorisé à l'occasion d'un retrait. Le code est différent. On voit que cette méthode a besoin d'accéder à l'attribut `solde` de la classe `Compte`. Or celui-ci avait été déclaré `private`, ce qui signifie qu'on ne peut l'utiliser que dans la classe `Compte`. Au moment de la compilation, cela provoque l'erreur suivante :

```
> javac CompteDecouvert.java
CompteDecouvert.java:19: error: solde has private access in Compte
        }else if (solde - montant < -decouvertAutorise){
                  ^
CompteDecouvert.java:22: error: solde has private access in Compte
            solde = solde - montant;
            ^
CompteDecouvert.java:22: error: solde has private access in Compte
```

```
            solde = solde - montant;
                    ^
3 errors
```

Pour pouvoir écrire notre sous-classe, il faut modifier la classe `Compte` en changeant la déclaration de l'attribut `solde` : il faut le déclarer `protected` au lieu de `private`.

4 Instances de la sous-classe

La sous-classe étant définie, on peut l'utiliser pour créer de nouveaux objets. Ceux-ci contiendront les attributs et méthodes hérités de `Compte`, plus l'attribut et les méthodes définis dans `CompteDecouvert`.

```
public class TesteCompteDecouvert{
    public static void main(String[] args){
        CompteDecouvert cptdec = new CompteDecouvert("lulu",501);
        cptdec.deposer(200);
        System.out.println(cptdec);
        System.out.println("Decouvert␣autorise:␣" + cptdec.getDecouvert());
        System.out.println("Solde:␣" + cptdec.getSolde());
        try{
            cptdec.retirer(1500);
        }catch(IllegalArgumentException exc){
            System.out.println("Erreur:␣" + exc.getMessage());
        }
    }
}
```

On voit dans ce code qu'on peut utiliser les méthodes `getSolde` et `deposer` bien qu'on ne les ait pas définis dans la classe. Ils sont néanmoins utilisable sur l'objet créé. On utilise également la méthode `getDecouvert` qui est nouvelle et `retirer` qui a été redéfinie. C'est bien évidemment la nouvelle version, celle de la sous-classe, qui sera exécutée.

```
> java TesteCompteDecouvert
Titulaire: lulu numéro: 501 solde: 200.0
Decouvert autorise: 500.0
Solde: 200.0
Erreur: Dépassement du découvert autorisé
```

Le message d'erreur qui s'affiche montre bien que c'est la nouvelle version de la méthode qui s'exécute puisque ce message n'existe pas dans la classe `Compte`.

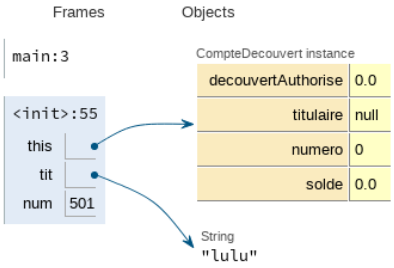
Il y a un défaut que révèle cette exécution : c'est que le découvert autorisé n'apparaît pas à l'écran lors de l'exécution de l'instruction `System.out.println(cptdec);`.

Pour cette raison, il est judicieux de redéfinir également la méthode `toString` dans la sous-classe.

5 Exécution dans Pythontutor

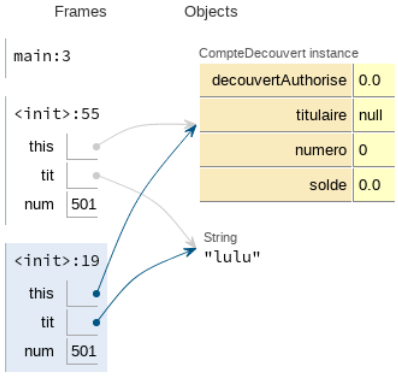
Exécutons le programme dans Pythontutor. Voyons se qui se passe à la création de l'objet. L'objet apparaît dans le tas lors de l'exécution de l'instruction `CompteDecouvert cptdec = new`

`CompteDecouvert("lulu",501);` Il y a appel au constructeur de la classe `CompteDecouvert`. Voilà l'image de la mémoire à ce moment-là.

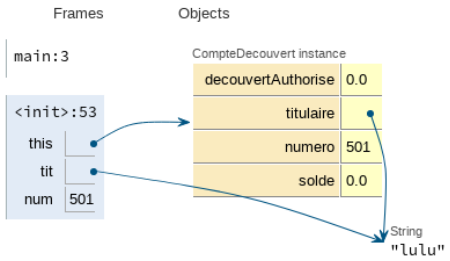


On voit que l'objet comporte quatre attributs : les trois définis dans `Compte` et qui ont été hérité dans `CompteDecouvert` plus `decouvertAutorise` qui est défini dans la sous-classe.

Un pas de plus dans l'exécution du programme va être l'exécution de l'instruction `super(tit,num);` qui est dans le code du constructeur. Cette instruction appelle le constructeur de la super-classe. Pythontutor affiche alors l'état suivant.

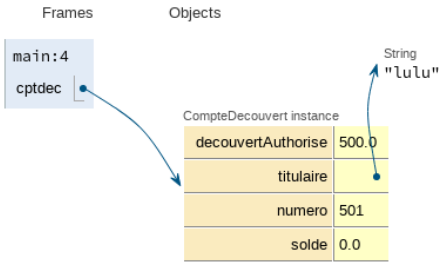


On voit dans la pile deux mémoires privées pour les deux constructeurs qui s'exécutent, le plus haut est celui de `CompteDecouvert`, le plus bas celui de `Compte`. On voit que les deux s'appliquent au même objet par le fait que le pseudo-attribut `this` contient une référence au même objet dans les deux mémoires privées. Le code du constructeur de `Compte` va s'exécuter pour initialiser les deux variables d'instance `titulaire` et `numero`. Après la fin de l'exécution du constructeur de `Compte`, on a l'état suivant.



On voit que le constructeur a initialisé les attributs `titulaire` et `numero` en utilisant les valeurs des paramètres `tit` et `num`.

Après la fin de l'exécution du constructeur de `CompteDecouvert`, on a l'état suivant où les variables d'instance ont leur valeur. La valeur de `DecouvertAutorise` a été initialisée par le constructeur de la sous-classe.



6 Hiérarchie d'héritage

Une classe ne peut hériter que d'une seule classe mère. En revanche, plusieurs classes peuvent hériter d'une même classe mère. Cela signifie qu'une classe peut avoir plusieurs classes filles.

Par ailleurs, une classe peut hériter d'une classe et avoir une ou plusieurs classes qui héritent d'elle.

Les relations d'héritage définissent une hiérarchie, comme un arbre généalogique où les filles apparaissent en-dessous des mères. La hiérarchie correspondant à notre exemple est la suivante.

Compte

|

CompteDecouvert

Par ailleurs, quand une classe ne comporte pas de clause `extends`, elle hérite quand même d'une classe mère par défaut : la classe prédéfinie `Object`. Cette classe comporte notamment les méthodes `toString` et `equals`, ce qui fait que tous les classes Java que l'on peut écrire ont ces deux méthodes par héritage. Donc la hiérarchie des classes de notre exemple complétée est :

Object

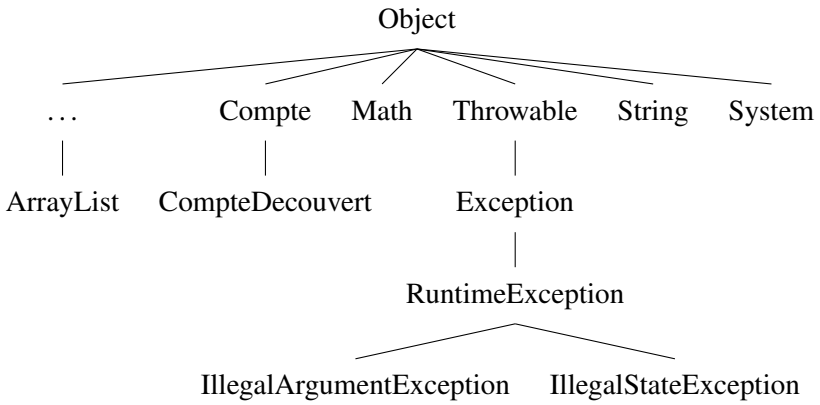
|

Compte

|

CompteDecouvert

Toutes les classes Java prédéfinies ou non sont dans une hiérarchie unique qui a la classe `Object` à la racine. Voici un extrait de cet arbre d'héritage.



7 Intérêt de l'héritage

Pour l'instant nous voyons un intérêt à l'héritage : c'est de pouvoir réutiliser les définitions d'attributs et de méthodes d'une classe dans une autre classe sans avoir à dupliquer le code. Cela ouvre la voie à des définitions de classes par raffinement successifs.

L'intérêt de l'héritage ne s'arrête pas là. Nous le verrons ultérieurement dans une séquence consacrée au typage des programmes comportant de l'héritage.