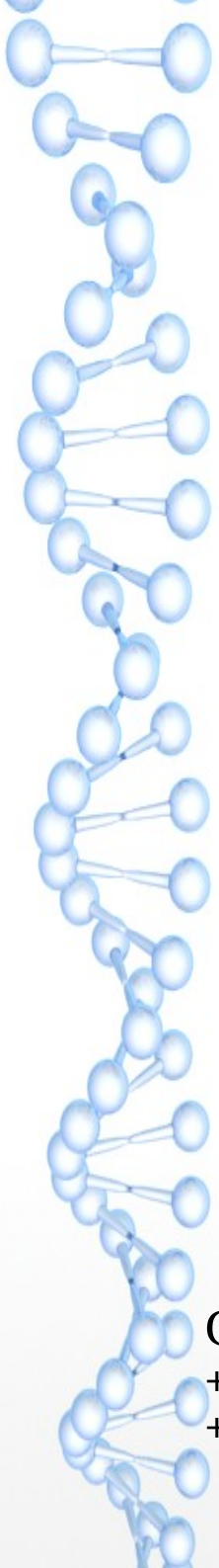

Kabola

Formation Développeur Java SE

Manuel de formation

Ce manuel n'est pas un cours complet

Contact
+ 242 06 602 22 22
+ 242 05 383 45 45

Module 1
Section 3

Kabola

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

Concept d'héritage

L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution.
Elle définit une relation entre deux classes :

- une classe mère ou super classe
- une classe fille ou sous classe qui hérite de sa classe mère

Principe de l'héritage

Grâce à l'héritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parente et peuvent les étendre. Les sous classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge.

L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super classes et de sous classes. Une classe qui hérite d'une autre est une sous classe et celle dont elle hérite est une super classe. Une classe peut avoir plusieurs sous classes. Une classe ne peut avoir qu'une seule classe mère : il n'y a pas d'héritage multiple en Java.

Object est la classe parente de toutes les classes en Java. Toutes les variables et méthodes contenues dans Object sont accessibles à partir de n'importe quelle classe car par héritages successifs toutes les classes héritent d'Object.

Kabola

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

L’enchaînement de références à des variables et à des méthodes

```
System.out.println("bonjour");
```

Deux classes sont impliquées dans l'instruction : System et PrintStream. La classe System possède une variable nommée out qui est un objet de type PrintStream. Println() est une méthode de la classe PrintStream. L'instruction signifie : « utilise la méthode println() de la variable out de la classe System ».

La surchage de méthodes

La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents. Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments.
La signature d'une méthode comprend le nom de la classe, le nom de la méthode et les types des paramètres.

Kabola

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

Concept d'héritage

L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution.
Elle définit une relation entre deux classes :

- une classe mère ou super classe
- une classe fille ou sous classe qui hérite de sa classe mère

Principe de l'héritage

Grâce à l'héritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parente et peuvent les étendre. Les sous classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge.

L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super classes et de sous classes. Une classe qui hérite d'une autre est une sous classe et celle dont elle hérite est une super classe. Une classe peut avoir plusieurs sous classes. Une classe ne peut avoir qu'une seule classe mère : il n'y a pas d'héritage multiple en Java.

Object est la classe parente de toutes les classes en Java. Toutes les variables et méthodes contenues dans Object sont accessibles à partir de n'importe quelle classe car par héritages successifs toutes les classes héritent d'Object.

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

La mise en œuvre de l'héritage

- **class Fille extends Mere { ... }**

L'accès aux propriétés héritées

Les variables et méthodes définies avec le modificateur d'accès public restent publiques à travers l'héritage et toutes les autres classes.

Une variable d'instance définie avec le modificateur private est bien héritée mais elle n'est pas accessible directement mais via les méthodes héritées.

Si l'on veut conserver pour une variable d'instance une protection semblable à celle assurée par le modificateur private, il faut utiliser le modificateur protected. La variable ainsi définie sera héritée dans toutes les classes filles qui pourront y accéder librement mais ne sera pas directement accessible hors de ces classes.

La redéfinition d'une méthode héritée

La redéfinition d'une méthode héritée doit impérativement conserver la déclaration de la méthode parente

La redéfinition d'une méthode héritée doit impérativement conserver la déclaration de la méthode parente.

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

Le polymorphisme

Le polymorphisme est la capacité, pour un même message de correspondre à plusieurs formes de traitements selon l'objet auquel ce message est adressé. La gestion du polymorphisme est assurée par la machine virtuelle dynamiquement à l'exécution.

Le polymorphisme est la faculté d'une classe à prendre plusieurs formes. Si une classe et ses sous-classes implémentent différemment un même message, les objets des sous-classes prennent des formes différentes pour ce message du point de vue de leur superclasse.

Kabola

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

Les interfaces

Avec l'héritage multiple, une classe peut hériter en même temps de plusieurs super classes. Ce mécanisme n'existe pas en Java. Les interfaces permettent de mettre en œuvre un mécanisme de remplacement.

Une interface est un ensemble de constantes et de déclarations de méthodes correspondant un peu à une classe abstraite.

Déclaration d'une interface

```
[public] interface nomInterface [extends nomInterface1, nomInterface2 ... ] {
    // insérer ici des méthodes ou des champs static

}
```

Implémentation d'une interface

```
Modificateurs class nomClasse [extends superClasse] [implements nomInterface1, nomInterface
2, ...] {
    //insérer ici des méthodes et des champs

}
```

Module 1 : CONCEPTS DE BASES

Section 3 : Programmation Orienté Objet partie 2

Les interfaces

Une interface peut être d'accès public ou package.

- Si elle est publique, toutes ses méthodes sont implicitement publiques même si elles ne sont pas déclarées avec le modificateur public.
- Si elle est d'accès package, il s'agit d'une interface d'implémentation pour les autres classes du package et ses méthodes ont le même accès package : elles sont accessibles à toutes les classes du packages.

Les seules variables que l'on peut définir dans une interface sont des variables de classe qui doivent être constantes : elles sont donc implicitement déclarées avec le modificateur static et final même si elles sont définies avec d'autres modificateurs.