valtech_

Memento

Programmation Java

Table of Contents

La classe

Une classe est le support de l'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité. Une classe est une description abstraite d'un objet. Les fonctions qui opèrent sur les données sont appelées des méthodes. Instancier une classe consiste à créer un objet à partir de son modèle (i.e. sa classe).

Java est un langage orienté objet : tout appartient à une classe sauf les variables de type primitives.

Pour accéder à une classe il faut en déclarer une instance de classe ou objet.

Une classe comporte sa déclaration, des variables et la définition de ses méthodes.

1. La syntaxe d'une classe

```
ClassModifiers class ClassName [extends SuperClass] [implemenents Interfaces]

{

// Insérer ici les champs et les méthodes

}
```

Les modificateurs de classe (ClassModifiers) sont :

Modificateur	Role
abstract	la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.
final	la classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées finales ne peuvent donc pas avoir de classes filles.
private	la classe n'est accessible qu'à partir du fichier où elle est définie
public	La classe est accessible partout

Le mot clé **extends** permet de spécifier une superclasse éventuelle : ce mot clé permet de préciser la classe mère dans une relation d'héritage.

Le mot clé **implements** permet de spécifier une ou des interfaces que la classe implémente. Cela permet de récupérer quelques avantages de l'héritage multiple.

L'ordre des méthodes dans une classe n'a pas d'importance. Si dans une classe, on rencontre d'abord la méthode A puis la méthode B, B peut être appelée sans problème dans A.

```
public abstract class Document {

}



public class Book extends Document{



}
```

2. Les objets

Les objets contiennent des attributs et des méthodes. Les attributs sont des variables ou des objets nécessaires au fonctionnement de l'objet.

En Java, une application est un objet. La classe est la description d'un objet. Un objet est une instance d'une classe. Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.

La création d'un objet : instancier une classe

Il est nécessaire de définir la déclaration d'une variable ayant le type de l'objet désiré.

La déclaration est de la forme : *nom_de_classe nom_de_variable*

```
Book book;

String info;
```

L'opérateur new se charge de créer une instance de la classe et de l'associer à la variable

```
Book = new Book() ;
```

Il est possible de tout réunir en une seule déclaration

```
Book book = new Book();
```

Chaque instance d'une classe nécessite sa propre variable. Plusieurs variables peuvent désigner un même objet.

En Java, tous les objets sont instanciés par allocation dynamique. Dans l'exemple, la variable homme contient une référence sur l'objet instancié (contient l'adresse de l'objet qu'elle désigne : attention toutefois, il n'est pas possible de manipuler ou d'effectuer des opérations directement sur cette adresse comme en C).

Si femme désigne un objet de type Humain, l'instruction femme = homme ne définit pas un nouvel objet mais le même.

La durée de vie d'un objet

Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.

La durée de vie d'un objet passe par trois étapes :

- la déclaration de l'objet et l'instanciation grâce à l'opérateur new

```
Book book = new Book();
```

- l'utilisation de l'objet en appelant ces méthodes
- la suppression de l'objet : elle est automatique en Java grâce à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector). Il n'existe pas d'instruction delete comme en C++.

Les propriétés ou attributs

Les données d'une classe sont contenues dans des variables nommées propriétés ou attributs. Ce sont des variables qui peuvent être des variables d'instances, des variables de classes ou des constantes.

```
public class abstract Document {

String title;

Date creationDate;

}

public class Book extends Document {

String author;

String editor;

int nbPages;

}
```

Les constructeurs

La déclaration d'un objet est suivie d'une sorte d'initialisation par le moyen d'une méthode particulière appelée constructeur pour que les variables aient une valeur de départ. Elle n'est systématiquement invoquée que lors de la création d'un objet.

Le constructeur suit la définition des autres méthodes excepté que son nom doit obligatoirement correspondre à celui de la classe et qu'il n'est pas typé.

On peut surcharger un constructeur.

La définition d'un constructeur est facultative. Si elle n'est pas définie, la machine virtuelle appelle un constructeur par défaut vide créé automatiquement.

Il existe plusieurs manières de définir un constructeur :

- le constructeur simple : ce type de constructeur ne nécessite pas de définition explicite : son existence découle automatiquement de la définition de la classe.

```
public Book() {}
```

- le constructeur avec initialisation fixe : il permet de créer un constructeur par défaut

```
public Book() {

nbPages = 0;

}
```

- le constructeur avec initialisation des variables : pour spécifier les valeurs de données à initialiser on peut les passer en paramètres au constructeur

```
public Book(int value) {

nbPages = value;

}
```

La variable this

Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. *this* est un objet qui est égale à l'instance de l'objet dans lequel il est utilisé.

```
public Book(int nbPages) {

nbPages = nbPages; // ambiguë

}
```

Il est préférable d'écrire

```
public Book(int nbPages) {

this.nbPages = nbPages;

}
```

This est aussi utilisé quand l'objet doit appeler une méthode en se passant lui même en paramètre de l'appel

Les références et la comparaison d'objets

Les variables de type objet que l'on déclare ne contiennent pas un objet mais une référence vers cet objet. Lorsque l'on écrit obj1 = obj2 (obj1 et obj2 sont des objets), on copie la référence de l'objet obj2 dans obj1 : obj1 et obj2 pointent sur le même objet.

L'opérateur == compare ces références.

Deux objets avec des propriétés identiques sont deux objets distincts :

```
Book b1 = new Book(100);

Book b2 = new Book(100);

if (b1 == b1) { ... } // vrai

if (b1 == b2) { ... } // faux
```

Pour comparer l'égalité des variables de deux instances, il faut munir la classe d'un méthode à cet effet : la méthode ***equals*** héritée de ***Object***.

Pour s'assurer que deux objets sont de la même classe, il faut utiliser la méthode ***getClass***() de la classe ***Object*** dont toutes les classes héritent.

```
public class Book extends Document {

...

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Book otherBook = (Book)obj;

return super.equals(obj) && otherBook.author.equals(author) && otherBook.title.equals(title);

}

}
```

3. Les mots clés qui gèrent la visibilité des entités

De nombreux langages orientés objet introduisent des attributs de visibilité pour réglementer l'accès aux classes et aux objets, aux méthodes et aux données.

Il existe 3 modificateurs qui peuvent être utilisés pour définir les attributs de visibilité des entités (classes, méthodes ou attributs) : ***public, private et protected.***

Leur utilisation permet de définir des niveaux de protection différents (présentés dans un ordre croissant de niveau de protection offert) :

Modificateur	Role
public	Une variable, méthode ou classe déclarée *public* est visible par tous les autres objets. Une seule classe « *public* » est permise par fichier et son nom doit correspondre à celui du fichier. Dans la philosophie orientée objet aucune donnée d'une classe ne devrait être déclarée publique : il est préférable d'écrire des méthodes pour la consulter et la modifier
par défaut package friendly	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une entité (classe, méthode ou variable) d'être visible par toutes les classes se trouvant dans le même package.
protected	Si une méthode ou une variable est déclarée *protected*, seules les méthodes présentes dans le même package que la classe ou ses sous classes pourront y accéder. On ne peut pas qualifier une classe avec *protected*.
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe : ils ne peuvent être modifiés que par des méthodes définies dans la classe prévues à cet effet. Les méthodes déclarées *private* ne peuvent pas être en même temps déclarées *abstract* car elles ne peuvent pas être redéfinies dans les classes filles.

```
public class abstract Document {

private String title;

private Date creationDate;

}

public class Book extends Document {

private String author;

private String editor;

private int nbPages;

...

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Book otherBook = (Book)obj;

return super.equals(obj) && otherBook.author.equals(author) && otherBook.title.equals(title);

}

}
```

Le mot clé static

Le mot clé *static* s'applique aux variables et aux méthodes.

Les variables d'instance sont des variables propres à un objet. Il est possible de définir une variable de classe qui est partagée entre toutes les instances d'une même classe : elle n'existe donc qu'une seule fois en mémoire. Une telle variable permet de stocker une constante ou une valeur modifiée tour à tour par les instances de la classe.

Elles ne sont définies qu'une seule fois quel que soit le nombre d'objets instanciés de la classe. Leur déclaration est accompagnée du mot clé ***static***

```
public class Book extends Document {

...

private static int nbCopies = 0;

}
```

Ces variables appartiennent à la classe entière et non à une de ses instances. Il est ainsi possible de l'utiliser de la manière suivante :

```
int nb = Book.nbCopies;
```

Ce type de variable est utile pour par exemple compter le nombre d'instanciation de la classe qui est faite.

Une méthode *static* est une méthode qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe. Ces méthodes peuvent être utilisées sans instancier un objet de la classe. Les méthodes ainsi définies peuvent être appelées avec la notation classe.methode() au lieu de objet.methode() : la première forme est fortement recommandée pour éviter toute confusion.

Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable d'instance à partir d'une méthode de classe statique.

Le mot clé final

Le mot clé final s'applique aux variables de classe ou d'instance, aux méthodes et aux classes. Il permet de rendre l'entité sur laquelle il s'applique non modifiable une fois qu'elle est déclarée pour une méthode ou une classe et initialisée pour une variable.

Une variable qualifiée de *final* signifie que la valeur de la variable ne peut plus être modifiée une fois que celle-ci est initialisée. On ne peut pas déclarer de variables finales locales à une méthode.

```
public class Book extends Document {

...

private final int isbn;

public Book(int isbn){

this.isbn = isbn;

}

}
```

Une fois la variable déclarée final initialisée, il n'est plus possible de modifier sa valeur. Une vérification est opérée par le compilateur.

Les constantes sont qualifiées avec ***final et static***.

```
public static final float PI = 3.141f;
```

Une méthode déclarée *final* ne peut pas être redéfinie dans une sous classe. Une méthode possédant le modificateur *final* pourra être optimisée par le compilateur car il est garanti qu'elle ne sera pas sous classée.

Lorsque le modificateur *final* est ajouté à une classe, il est interdit de créer une classe qui en hérite.

Le mot clé abstract

Le mot clé *abstract* s'applique aux méthodes et aux classes.

Abstract indique que la classe ne pourra être instanciée telle quelle. De plus, toutes les méthodes de cette classe ne sont pas forcément implémentées et devront être redéfinies par des méthodes complètes dans ses sous classes.

Abstract permet de créer une classe qui sera une sorte de moule. Toutes les classes dérivées pourront profiter des méthodes héritées et n'auront à implémenter que les méthodes déclarées abstract.

Une méthode abstraite est une méthode déclarée avec le modificateur *abstract* et sans corps. Elle correspond à une méthode dont on veut forcer l'implémentation dans une sous classe. L'abstraction permet une validation du codage : une sous classe sans le modificateur *abstract* et sans définition explicite d'une ou des méthodes abstraites génère une erreur de compilation.

Une classe est automatiquement abstraite dès qu'une de ses méthodes est déclarée abstraite. Il est possible de définir une classe abstraite sans méthodes abstraites.

```
public abstract class Document {

public abstract void borrowOneDocument();

}



public class Book extends Document{

public void borrowOneDocument(){

// Comportement de la méthode

}

}
```

Les tableaux

Ils sont dérivés de la classe Object : il faut utiliser des méthodes pour y accéder dont font partie des messages de la classe. Le premier élément d'un tableau possède l'indice 0.

1. La déclaration des tableaux

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

```
int[] tableau = new int[2];

ou

int tableau[] = new int[2];
```

Les tableaux peuvent comporter des types primitifs ou des types objets.

```
Document[] docs = new Document[2];
```

2. L'initialisation d'un tableau

Il est possible d'initialiser directement le tableau en fournissant entre {} les valeurs désirées. Cela évite de préciser une taille au tableau, celle-ci est implicite.

```
int[] tableau = {10,34,52};
```

Si le contenu du tableau est de type objet, la valeur par défauts sera null dans les cases.

Pour créer des valeurs différentes de null, il faut initialiser chaque case/

```
Book[] books = new Book[2];



books[0] = new Book();

books[1] = new Book();
```

3. Le parcours d'un tableau

Il y a deux solutions pour parcourir les tableaux :

- en utilisant la propriété length des tableaux qui fournit le nombre d'éléments du tableau et une boucle for

```
for(int i = 0; i < books.length ; i++){ ... }
```

- Depuis le JDK 1.5, il est possible d'utiliser un foreach pour parcourir les éléments du tableau.

```
for(Book b : books){ ... }
```

Héritage, polymorphisme

1. L'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

Le principe de l'héritage

Grâce à l'héritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parent et peuvent les étendre. Les sous classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les re-déclarer 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.

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.

La mise en oeuvre de l'héritage

On utilise le mot clé ***extends*** pour indiquer qu'une classe hérite d'une autre. En l'absence de ce mot réservé associé à une classe, le compilateur considère la classe Object comme classe parent.

```
public class Book extends Document{

...

}
```

Pour invoquer une méthode d'une classe « *parent* », il suffit d'indiquer la méthode préfixée par super. Pour appeler le constructeur de la classe parent il suffit d'écrire « *super(paramètres)* » avec les paramètres adéquats.

En Java, il est obligatoire dans un constructeur d'une classe fille de faire appel explicitement ou implicitement au constructeur de la classe mère.

```
public abstract class Document{

private String title;

private Date creationDate;



public Document(String title){

this.title = title;

creationDate = new Date();

}

}



public class Book extends Document{

public Book(String title, String author,

String editor, int nbPages){

super(title);

this.author = author;

this.editor = editor;

this.nbPages = nbPages:

}

}
```

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***.

2. Le polymorphisme

Le polymorphisme correspond à la possibilité pour un opérateur ou une fonction d'être utilisable dans des contextes différents et d'avoir un comportement adapté à ses paramètres. Il s'agit d'un élément essentiel qui permet la réutilisation de programmes existants et l'extensibilité des applications.

C'est grâce au polymorphisme que des types définis comme spécialisation d'un même ancêtre vont pouvoir, si besoin est, exprimer leurs différences avec cet ancêtre et entre eux.

```
public class DVD extends Document{

private int duration ;

public DVD(String title, int duration ){

super(title) ;

this.duration = duration ;

}

}

…

Document[] tab = new Document[2];



//Possible de créer un tableau de Document comportant à la fois des Book //et des DVD

tab[0] = new Book(“Book”,”author”,”editor”, 100);

tab[1] = new DVD(“DVD”,120);



//Possible d'appeler de manière générique les méthodes

for(Document doc : tab){

doc.borrowOneDocument();

}
```

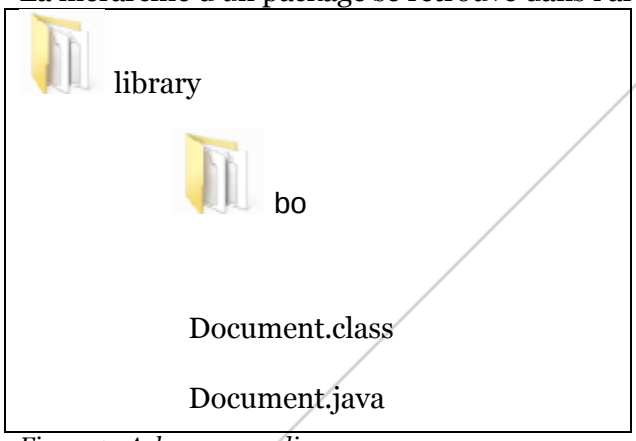
Les packages

1. La définition d'un package

En Java, il existe un moyen de regrouper des classes voisines ou qui couvrent un même domaine : ce sont les packages.

Pour réaliser un package, on écrit un nombre quelconque de classes dans plusieurs fichiers d'un même répertoire et au début de chaque fichier on met la directive ci-dessous ou nom–du–package doit être identique au nom du répertoire : ***package nomPackage;***

La hiérarchie d'un package se retrouve dans l'arborescence du disque dur :




```
package library.bo;

public abstract class Document{

...

}
```

Figure 1 : Arborescence disque

Remarque : Il est préférable de laisser les fichiers source .java avec les fichiers compilés .class

D'une façon générale, l'instruction package associe toutes les classes qui sont définies dans un fichier source à un même package.

Le mot clé package doit être la première instruction dans un fichier source et il ne doit être présent qu'une seule fois dans celui-ci (une classe ne peut pas appartenir à plusieurs packages).

2. L'utilisation d'un package

Pour utiliser ensuite le package ainsi créé, on l'importe dans le fichier : ***import nomPackage.*;***

```
package library.bo.dvd;


import library.bo.Document;//Solution en précisant la classe à importer


//OU


import library.bo.*;//permet d'accéder à toutes classes dans ce package


public class DVD extends Document{

...

}
```

Classes internes et anonymes

1. Les classes internes

Les classes internes (inner classes) sont une extension du langage Java. Ce sont des classes qui sont définies dans une autre classe. Les difficultés dans leur utilisation concernent leur visibilité et leur accès aux membres de la classe dans laquelle elles sont définies.

```
public class Book extends Document {

...

class Roman {

}

}
```

Les classes internes sont particulièrement utiles pour :

- permettre de définir une classe à l'endroit ou une seule autre en a besoin définir des classes de type adapter (pour traiter des événements émis par les interfaces graphiques)
- définir des méthodes de type callback d'une façon générale

Il est possible d'imbriquer plusieurs classes internes. Java ne possèdent pas de restrictions sur le nombre de classes qu'il est ainsi possible d'imbriquer. En revanche une limitation peut intervenir au niveau du système d'exploitation en ce qui concerne la longueur du nom du fichier .class généré pour les différentes classes internes.

Le nom de la classe interne utilise la notation qualifiée avec le point préfixé par le nom de la classe principale. Ainsi, pour utiliser ou accéder à une classe interne dans le code, il faut la préfixer par le nom de la classe principale suivi d'un point.

Pour pouvoir utiliser une variable de classe dans une classe interne, il faut la déclarer dans sa classe englobante.

2. Les classes anonymes

Les classes internes anonymes (anonymous inner–classes) sont des classes internes qui ne possèdent pas de nom. Elles ne peuvent donc être instanciées qu'à l'endroit où elles sont définies.

Ce type de classe est très pratique lorsqu'une classe doit être utilisée une seule fois : c'est par exemple le cas d'une classe qui doit être utilisée comme un callback.

Une syntaxe particulière de l'opérateur new permet de déclarer et instancier une classe interne :

```
new classe_ou_interface () {

// définition des attributs et des méthodes de la classe interne

}
```

Cette syntaxe particulière utilise le mot clé new suivi d'un nom de classe ou interface que la classe interne va respectivement étendre ou implémenter. La définition de la classe suit entre deux accolades.

Une classe interne anonyme peut soit hériter d'une classe soit implémenter une interface mais elle ne peut pas explicitement faire les deux.

Si la classe interne étend une classe, il est possible de fournir des paramètres entre les parenthèses qui suivent le nom de la classe.

Les classes internes anonymes qui implémentent une interface héritent obligatoirement de classe Object. Comme cette classe ne possèdent qu'un constructeur sans paramètre, il n'est pas possible lors de l'instanciation de la classe interne de lui fournir des paramètres.

Les classes anonymes sont un moyen pratique de déclarer un objet sans avoir à lui trouver un nom. La contre- partie est que cette classe ne pourra être instanciée dans le code qu'à l'endroit où elle est définie : elle est déclarée et instanciée en un seul et unique endroit.