JAVA
FONDAMENTAUX

Présenté par :

Xavier TABUTEAU

Les classes

Java met en œuvre les concepts suivants :

- Les classes
- Les objets
- L’encapsulation
- Les JavaBeans
- L’héritage
- L’abstraction
- Le polymorphisme

Les classes

Une classe est le modèle (un moule) qui va recevoir essentiellement les attributs (variables) et méthodes (fonctions) de l'objet. On peut créer autant d'objet qu'on veut à partir d'une classe. Ils seront indépendants.

Dans Java il est possible de définir plusieurs classes dans un même fichiers suffixé par .java mais, parmi ces classes, une seule sera définie comme « public », et c'est elle qui donnera le nom au fichier. Une classe a un nom commençant par une majuscule.

Client.java

```
1  package gestion;
2  public class Client {
3      private String nomclient;
4      private Integer statut;
5      public Boolean creerClient(String nomclient, Integer statut) {
6          this.nomclient=nomclient;
7          this.statut=statut;
8          return true;
9      }
10     public Boolean changerStatut(Integer statut) {
11         if (this.statut==0) {
12             return false;
13         }
14         this.statut=statut;
15         return true;
16     }
17 }
18 class Adresse{
19     public String adresse;
20 }
21
```

La visibilité

On trouve 3 mots-clés pour la visibilité (accessibilité) des attributs et méthodes mais 4 visibilités différentes.

- public : La portée est totale, même en dehors du package.
- private : Opposé de public, inaccessible depuis l'extérieur d'une classe.
- protected : Accès depuis toutes les classes du même package ou des classes qui héritent de celle-ci (sous-classes).
- « friendly » : L'accès est par défaut depuis toutes les classes du même package mais pas aux sous-classes (Il est appelé friendly mais ne s'écrit pas).

Une classe ne peut avoir que la visibilité « public » ou « friendly ».

```
2  public class Client {
3      private String nomclient;
4      private Integer statut;

23         Client monclient = new Client();
24         monclient.nomclient="Sarl";
```

Erreur de compilation !!!

Les constructeurs

Le constructeur est une méthode particulière permettant de créer un objet. En cas de passage de paramètres dans le constructeur, il permet aussi de l'initialiser avec ceux-ci.

Un constructeur est une méthode spéciale qui porte obligatoirement le nom de la classe dans laquelle il est défini.

Il n'a pas type de retour.

Il n'est pas obligatoire, mais très utile.

Il est appelé automatiquement lorsqu'on demande la création d'un objet de la classe de ce constructeur (utilisation du mot clé « new »).

Il peut y avoir plusieurs constructeurs dans une classe. Ils se différencient par le nombre et type de paramètres passés. Comme une surcharge de méthode classique.

Il peut être appelé depuis un autre constructeur de façon explicite avec le mot-clé « this ». Comme avec des méthodes surchargées classique.

Remarque : Il n'existe pas vraiment de destructeur. En fait, la mémoire est gérée par le Garbage Collector. Quand un objet n'est plus référencé en mémoire le Garbage Collector libère la mémoire que prenait l'objet.

Constructeurs.java

This

Le mot-clé « this » se réfère à l'instance de l'objet en cours.

```
2  public class Client {
3      private String nomclient;
4      private Integer statut;
5      private Adresse adresse;
6      public Boolean creerClient(String nomclient, Integer statut, String adresse) {
7          this.nomclient=nomclient;
8          this.statut=statut;
9          this.adresse = new Adresse();
10         this.adresse.setAdresse(adresse);
11         return true;
12     }
13     public Boolean changerStatut(Integer statut) {
14         if (this.statut==0) {
15             return false;
16         }
17         this.statut=statut;
18         return true;
19     }
20     public String getAdresse() {
21         return this.adresse.getAdresse();
22     }
23 }
24 class Adresse{
25     private String adresse;
26     public String getAdresse() {
27         return adresse;
28     }
29     public void setAdresse(String adresse) {
30         this.adresse = adresse;
31     }
```

Déclaration de la propriété de type Adresse

Stockage de l'adresse dans l'objet de type Adresse, nommé adresse

Accesseur, ou getter, permettant l'accès au contenu de l'adresse via la propriété de type Adresse

Propriété adresse de la classe Adresse, avec ses deux accesseurs.

Classes.java

L’encapsulation

L’encapsulation en Java est un terme qui recouvre une chose simple : Protéger les données d’un objet. Cela permet de créer un couplage faible au lieu d’un couplage fort entre objets et classes. Le couplage faible est recommandé car il permet une meilleure maintenabilité du code.

Une classe peut contenir des données de type public et private. Si on déclare toutes les propriétés en public, alors l’encapsulation est inutile car nous aurons un couplage fort.

```
1  package nonencapsule;
2  public class Client{
3      public void crediterCompte(double credit) {
4          CompteBancaire cb = new CompteBancaire();
5          cb.crediter(100.12);
6          cb.valeur=200;
7      }
8  }
9  class CompteBancaire {
10     public double valeur;
11     public double crediter(double credit) {
12         this.valeur+=credit;
13         return this.valeur;
14     }
15 }
16
```

Accessible directement !

Non protégée !

L’encapsulation

Les accesseurs (ou Getters et Setters en anglais) sont des méthodes qui permettent de ne pas accéder directement à la variable de la classe qui elle, est privée. C’est donc un couplage faible. Cela permet de contrôler les accès aux attributs de la classe.

```
1  package nonencapsule;
2  public class Client{
3      public void crediterCompte(double credit) {
4          CompteBancaire cb = new CompteBancaire();
5          cb.crediter(100.12);
6      }
7  }
8  class CompteBancaire {
9      private double valeur;
10     public double crediter(double credit) {
11         this.valeur+=credit;
12         return this.valeur;
13     }
14     public double getValeur() {
15         return valeur;
16     }
17 /*  public void setValeur(double valeur) {
18         this.valeur = valeur;
19     }*/
20 }
21
```

Protégée !

Accesseur pour accéder à la valeur → **Encapsulation**

GettersSetters.java

JavaBean

C’est une représentation unique d’une entité fonctionnelle utilisable à divers endroits du programme.

Exemple :

- Un client
- Une commande
- Un article

Acronymes similaires pour dire JavaBean :

POJO (Plain Old Java Object)
DTO (Data Transfer Object)
DAO (Data Access Object)
...

JavaBean

Un JavaBean est une classe qui doit respecter les conventions suivantes :

- Implémenter l'interface Serializable.
- Proposer un constructeur sans paramètre.
- Disposer d'accesseur publiques pour les attributs private (getters et setters).
- Classe non déclarée « final ».

JavaBean

Exemple d'utilisation :

- Classe métier ArticleService : réalisera des opérations sur un article (création, modification, suppression, …)

- Les méthodes de cette classe travaillerons sur le JavaBean Article.

Surcharge

La surcharge en Java c'est la capacité d'une classe à accepter d'avoir des méthodes avec le même nom. Il est possible d'utilisé le mot clé this pour appeler une méthode surchargée dans une autre.

Contrainte : différenciation sur le nombre et/ou la nature des types qui sont déclarés pour cette méthode.

Attention : le type de retour ne peut servir de discriminant !

Exemple de surcharge dans les méthodes du JDK :

La méthode valueOf de la classe String

Le mot clé static

static lie un attribut ou une méthode à la classe. Cela à pour utilité de pouvoir définir des propriétés au niveau global de la classe. Donc ces attributs ou méthodes sont des objets uniques. Modifier un attribut « static » d'une classe, modifie cet attribut pour tous les objets instanciés de cette classe. Par convention, on y accède via le nom de la classe au lieu du nom de l'objet.

Le mot clé final

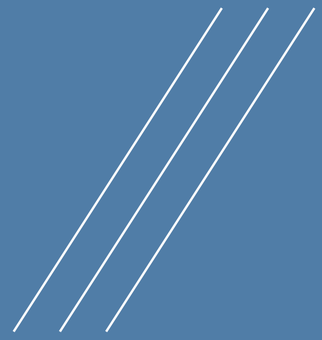
Ce mot clé s'utilise sur un attribut, une méthode ou une classe. Après initialisation sur un attribut celle-ci ne peut plus être changée ! Cela provoquerait une erreur de compilation. En résumé, ce mot clé permet de déclarer une constante. Par convention de nommage, les constantes doivent être écrites en majuscule.

Remarque : associé « static » et « final » est possible. Cela permet de créer une constante unique à la classe.

Le mot clé final appliqué sur une méthode, permet l'interdiction de la redéfinition de cette méthode dans le cas de l'héritage.
Le mot clé final appliqué sur une classe, permet l'interdiction de l'héritage depuis celle-ci.

Sera vu plus tard ;)

JAVA
FONDAMENTAUX



Présenté par
Xavier TABUTEAU