Concepts de base de la POO

Sommaire

1 Classe

La génération spontanée n'existe pas. La voiture que vous conduisez, la chaise sur laquelle vous êtes assis(e) ont été fabriquées. Tous ces objets proviennent d'une "fabrique" ou usine. Pour une première introduction à la notion de classe, ce dernier terme peut être retenu. Il existe ainsi des usines pour avions, bateaux, jouets, etc.
Une autre particularité des classes est le principe de regroupement. On évite dans une même usine de fabriquer à la fois des voitures et des bateaux, ou des jouets et des ordinateurs. Les objets fabriqués appartiennent donc à des catégories qui sont les classes en POO. Nous pouvons retenir qu'une classe est une sorte d'usine qui fabrique des objets qui ont des caractéristiques communes (les bateaux ont une coque, les voitures une carrosserie).
Nous avons vu lors de l'analyse, leur représentation graphique UML. Voyons maintenant précisément leur écriture en Java au travers d'exemples très simples.

Pour déclarer une classe Chat, il suffit d'utiliser le mot clé Class :

```
class Chat {
}
```

2 Propriétés

La classe Chat est ensuite complétée selon les besoins de chacun. Supposons que nous désirons juste caractériser les chats par leur nom, couleur et âge. Il faut alors ajouter ces caractéristiques ou propriétés à la classe.

```
class Chat {
      String caractere;
      String nom;
      String couleur;
      int age;
}
```

Toute propriété ou attribut doit impérativement être typée : entier pour l'âge et chaîne de caractères pour les autres dans notre exemple. Ces types sont simples. On parle de types élémentaires ou primitifs. Vous aurez l'occasion lors du développement d'utiliser des types plus riches notamment des types classes. Les propriétés définissent l'aspect structurel d'une classe.

3 Méthodes

Les méthodes représentent les actions que peuvent mener des objets issus d'une classe. Elles définissent l'aspect dynamique ou comportemental de la classe. Les chats peuvent dormir, manger, miauler… Ajoutons quelques-unes de ces méthodes.

```
class Chat {
      // propriétés
      String caractere;
      String nom;
      String couleur;
      int age;
      // méthodes
      void dormir(){
      }
      void manger(){
      }
      String miauler(){
            String leMiaulement = "miaou";
            return leMiaulement;
      }
}
```

Les méthodes qui ne renvoient aucune valeur correspondent à des procédures. Elles sont toujours précédées du mot clé void.
Les méthodes qui renvoient une valeur correspondent à des fonctions. Elles sont toujours précédées du type de la valeur renvoyée.
Les méthodes peuvent bien sûr contenir des paramètres qui doivent tous être typés.
Parmi les méthodes, certaines ont pour rôle de retourner les valeurs des propriétés et d'autres de les modifier. Elles sont nommées respectivement accesseurs et mutateurs.

```
class Chat {

      // propriétés
      ...

      // méthodes
     String getNomChat() {
       return nom;
     }
     void setNom(String vNom) {
       nom = vNom;
     }
     String getCouleur() {
       return couleur;
     }
     void setCouleur(String vCouleur) {
       couleur = vCouleur;
     }
     int getAge() {
       return age;
     }
     void setAge(int vAge) {
       age = vAge;
     }
      ...
     }
}
```

Rq : Le terme de services est parfois utilisé pour désigner les méthodes et celui d'opération pour l'implémentation des méthodes.

4 Accessibilité

La POO ajoute le principe d'accessibilité (visibilité) qui consiste à contrôler l'accès aux classes et à leurs membres (propriétés et méthodes). Les niveaux de visibilité sous Java sont les suivants :

4.1 Membres

- public : les membres sont utilisés sans aucune restriction par toute classe et en tout lieu. Pour protéger les propriétés de modifications non autorisées, il est cependant fortement recommandé de ne pas les déclarer publiques.
- private : c'est le niveau le plus élevé de restriction. Les propriétés et méthodes ne sont alors accessibles que depuis la classe elle-même. Concernant les propriétés, elles peuvent l'être en dehors de la classe mais uniquement par le biais de ses mutateurs et accesseurs déclarés publics.
- protected : les membres sont accessibles aux classes descendantes (voir la notion d'héritage plus loin) mais aussi aux classes non descendantes du même package.

4.2 Classes

- public : en java, les classes sont publiques par défaut. Il convient cependant de le préciser par le mot clé public. Son omission rend par ailleurs les classes inaccessibles en dehors de leur package.
- protected : ce niveau ne concerne que les classes internes. Ce sont des classes définies à l'intérieur d'une classe n'étant utilisées que par celle-ci.

D'autres mots clés permettent de caractériser les classes et leurs membres dans des domaines autres que celui de l'accessibilité tels que la sécurité ou l'optimisation.

```
public class Chat {
  // propriétés
  private String nom;
  private String couleur;
  private int age;

  // méthodes
  public String getNomChat() {
    return nom;
  }
  public void setNom(String nom) {
    this.nom = nom;
  }
  public String getCouleur() {
    return couleur;
  }
  public void setCouleur(String couleur) {
    this.couleur = couleur;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public void dormir(){
  }

  public void manger(){
  }
```

```
  public String miauler(){
    String leMiaulement = "miaou";
    return leMiaulement;
  }
}
```

5 Encapsulation

Le fait de regrouper au sein d'une même classe propriétés et méthodes, et de définir le niveau d'accessibilité constitue l'encapsulation, un des concepts essentiels de la POO. Seules sont visibles les méthodes publiques, ce qui renforce la sécurité des données et la maintenance des programmes. De plus, l'implantation de la méthode est masquée. Finalement avec le concept d'encapsulation, la classe ne présente qu'une interface composée des méthodes publiques et de leurs éventuels paramètres.
Si nous reprenons notre exemple, pour modifier l'âge d'un chat, il faut utiliser la méthode setAge(). En partant de ce principe, on peut l'appliquer à d'autres cas tel que le débit d'un compte qui ne peut se faire sans la méthode appropriée et habilitée. Les risques d'erreur sont ainsi réduits et la sécurité accrue. La modification du code traitant du miaulement d'un chat ou du débit d'un compte n'a que peu voire aucune incidence sur un programme puisque l'implémentation des méthodes est masquée. Exemple :

```
// méthode modifiée
  public String miauler(){
    // un chat qui n'arrête pas de miauler
    String leMiaulement = "miaou miaouuuu miaouuuu ...";
    return leMiaulement;
  }

// utilisation dans un programme
leChat.miauler();
```

6 Constructeur

Pour qu'une classe soit complète, il lui faut un constructeur. Il s'agit d'une méthode particulière qui permet de créer des objets. Si vous omettez de définir un constructeur, Java le fera pour vous au moment de la compilation du programme en créant un constructeur basique du type nomDelaClasse(). Ajoutons un constructeur à notre classe. Exemple :

```
public class Chat {
  // propriétés
  ...
  // constructeur
  public Chat(String leNom, String laCouleur, int sonAge){
    nom = leNom;
    couleur = laCouleur;
    age = sonAge;
  }
  // méthodes
  ...
}
```

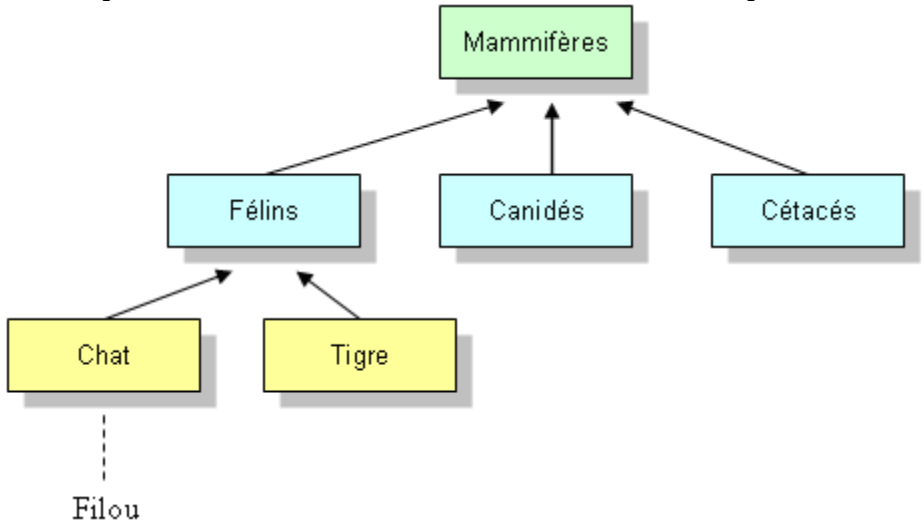
Le constructeur porte le même nom que la classe. Il est toujours public sinon comment les objets pourraient-ils être créés ? Le constructeur Chat() attend trois paramètres ce qui permet de créer un chat avec un nom, une couleur et un âge. Exemple dans un programme :

```
...
Chat leChat = new Chat("Filou", "rouquin", 2);
...
```

Rq : Certaines classes particulières n'ont pas besoin de constructeurs, ce sont les interfaces. Elles sont constituées uniquement de méthodes abstraites.

7 Héritage

L'héritage est un concept aisé à comprendre. Par exemple, nous héritons de nos parents certains traits physiques et peut-être certains comportements. Tel père tel fils, dit-on souvent. En POO, l'idée est la même. Si on transpose le concept d'héritage aux classes, cela revient à définir une classe générique dont dérivent d'autres classes. Pour la classe Chat, il est possible de la rattacher à la classe mammifère. Selon la partie du monde réel des animaux que l'on désire représenter, une structure de classes est créée. Exemple :



Tous les félins, canidés et cétacés héritent de la classe Mammifères des caractéristiques communes : animaux à sang chaud, la femelle allaite ses petits, etc. Filou fait partie d'une grande famille. Il va hériter des classes ancêtres toutes les propriétés et méthodes de celles-ci sauf celles déclarées privées. Pour en tenir compte, il faut revoir la définition de la classe Chat et ajouter celles des classes parentes. Pour faire simple, voici un exemple d'héritage limité aux classes Félin et Chat.

```
public class Felin {
  private String espece;
  private String couleur;
  public Felin(String espece, String couleur) {
    this.espece = espece;
    this.couleur = couleur;
  }
  // les mutateurs et accesseurs
  public String getCouleur() {
    return couleur;
  }
  public void setCouleur(String couleur) {
    this.couleur = couleur;
  }
  public String getEspece() {
    return espece;
  }
  public void setEspece(String espece) {
    this.espece = espece;
  }

}

public class Chat extends Felin {
```

```
  private String nom;
  private int age;
  public Chat(String vEspece, String vCouleur, String vNom, int vAge) {
    super(vEspece, vCouleur);
    nom = vNom;
    age = vAge;
  }
  // les mutateurs et accesseurs
  ...
  // les autres méthodes
  ...
}
```

Le mot clé extends indique que la classe dérivée (fille) Chat étend la classe de base (mère) Felin. L'appel du constructeur de la classe mère s'effectue avec le mot clé super.

8 Redéfinition des méthodes

Les méthodes de la classe mère peuvent être redéfinies dans les classes filles ou sous-classes. Il s'agit en fait d'une spécialisation. Dans le sens inverse, c'est une généralisation qui permet de factoriser les comportements communs. Par exemple, tous les mammifères se nourrissent mais les chats domestiques mangent des croquettes ce qui n'est pas forcément le cas des tigres. Voyons ce que cela donne avec nos classes :

```
public class Felin {
  ...
  public void manger(String vNourriture){
  }
}

public class Chat {
  ...
  public void manger(String vNourriture){
    System.out.println("Je mange " + vNourriture);
  }
}
```

Exemple dans un programme :

```
  ...
  leChat.manger("croquettes")
```

9 Polymorphisme

Dans une première approche, le polymorphisme correspond à la capacité d'un objet à choisir selon son type (sa classe) au moment de l'exécution (dynamiquement) la méthode qui correspond le mieux à ses besoins parmi ses propres méthodes ou celles des méthodes mères. Une autre particularité du polymorphisme est que toutes les méthodes concernées portent le même nom. Au sein d'une classe, elles se différencient obligatoirement par le nombre de paramètres ou le type de ces derniers.
En ajoutant à la structure de classes précédente, la classe Baleine, nous avons à définir ce que mange les baleines avec une méthode se nommant également manger(). Modifions aussi la classe Chat pour mettre en évidence le polymorphisme de méthodes. Exemple :

```
public class Baleine {
  ...
  public void manger(String vNourriture){
    System.out.println("Je mange du " + vNourriture + ". Qui suis-je ?");
  }
}

public class Chat {
```

```
  ...
  public String ronronner(){
    String leMiaulement = "ron ron ...";
    return leMiaulement;
  }
  public void manger(String vNourriture){
    System.out.println("Je mange des " + vNourriture);
    System.out.println("Quand j'ai bien mangé, je ronronne " +
ronronner());
  }
}
```

Exemple dans un programme :

```
  ...
  leChat.manger("croquettes");
  laBaleine.manger("plancton");
```

À l'exécution :

```
Je mange du plancton. Qui suis-je ?
Je mange des croquettes
Quand j'ai bien mangé, je ronronne ron ron ...
```

La deuxième forme de polymorphisme correspond à la capacité d'un objet à changer de type, autrement dit de classe. Cela consiste à sous-typer une propriété de la classe mère.