TP 5 :
Introduction à la programmation objet

Objectifs : Connaître les concepts de la POO.
Difficulté : Moyenne.

Dans le langage Java, la programmation est essentiellement Objet.
En java rien n'existe sans les Objets.
Tout est Objet !

A)Programmation Orientée Objet

De manière superficielle, le terme « orienté objet », signifie que l'on organise le logiciel comme une collection d'objets dissociés comprenant à la fois une structure de données (**attributs**) et un comportement (**méthodes**) dans une même entité (**encapsulation**).

Exemple : une voiture peut avoir une certaine couleur et en même temps possède un comportement qui sera le même pour toutes les autres voitures, comme accélérer.

Ce concept est différent de la programmation conventionnelle dans laquelle les structures de données et le comportement ne sont que faiblement associés.

1. Les Objets

Chaque objet a une identité et peut être distingué des autres. Deux pommes ayant les mêmes couleur, forme et texture demeurent des pommes individuelles; une personne peut manger l'une, puis l'autre. De la même façon, des jumeaux identiques sont deux personnes distinctes, même si elles se ressemblent.

Le terme identité signifie que les objets peuvent être distingués grâce à leur existence inhérente et non grâce à la description des propriétés qu'ils peuvent avoir.

Nous utiliserons l'expression instance d'objet pour faire référence à une chose précise, et l'expression classe d'objets pour désigner un groupe de choses similaires. En d'autres termes, deux objets sont distincts même si tous leurs attributs (nom, taille et couleur par exemple) ont des valeurs identiques (deux pommes vertes sont deux objets distincts).

2. Les Classes

La classification signifie que les objets ayant la même structure de donnée (attributs) et le même comportement (méthodes) sont regroupés en une classe.

Les objets d'une classe ont donc le même type de comportement et les mêmes attributs.

En groupant les objets en classe, on abstrait un problème.

Les définitions communes (telles que le nom de la classe et les noms d'attributs) sont stockées une fois par classe plutôt qu'une fois par instance.

Les méthodes peuvent être écrites une fois par classe, de telle façon que tous les objets de la classe bénéficient de la réutilisation du code.

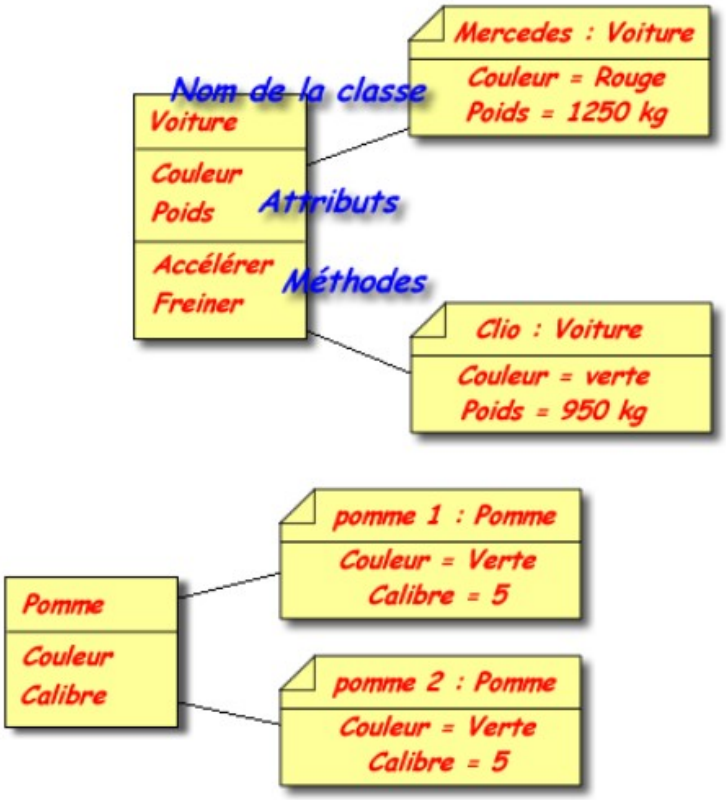
Par exemple, toutes les ellipses partagent les procédures de dessin, de calcul d'aire ou de test d'intersection avec une ligne.

Une **classe** est un modèle utilisé pour créer plusieurs objets présentant des caractéristiques communes.

Chaque objet possède ses propres valeurs pour chaque attribut mais partage noms d'attributs et méthodes avec les autres objets de la classe.

Par exemple une classe Voiture peut être définie comme ayant un des attributs couleur et comme une des méthodes freiner ; les objets associés à la classe voiture peuvent être : Mercedes rouge, Clio verte, etc.. Ces objets ont le même comportement (freiner).

On peut acheter un kilo de pommes vertes calibrées, chacune de ces pommes est un objet unique qui correspond à la classe Pomme avec comme attributs (couleur, calibre).



La relation qui existe entre la classe et les objets est la même que celle qui existe entre un type de données et des variables de ce type.

On dit que l'objet est une **instance** (variable) **de sa classe** (type).

Nota: Rappelons que la notion de variable informatique est totalement identique à la variable mathématique et que la notion de type correspond à l'ensemble de définition. De façon plus concrète, la variable correspond à un emplacement mémoire, et il est nécessaire de connaître son type pour déterminer sa taille

3. Les attributs

Un attribut est une valeur de donnée détenue par les objets de la classe. Couleur et Poids sont des attributs des objets relatifs à Voiture. Chaque attribut à une valeur pour chaque instance d'objet.

Par exemple l'attribut Couleur porte la valeur rouge dans l'objet Mercedes alors que pour l'objet Clio la valeur de l'attribut Couleur est verte.

Les instances peuvent avoir des valeurs identiques ou différentes pour un attribut donné. Chaque nom d'attribut est unique à l'intérieur d'une classe.

Ainsi, la classe Voiture et la classe Pomme peuvent avoir chacune un attribut Couleur.

Dans une classe, les **attributs** sont définis par des variables. Les attributs peuvent être considérés comme des variables globales pour chaque objet de cette classe. Comme pour toutes variables, il est nécessaire de connaître le **type** correspondant et c'est la classe de l'objet qui indique de quel type d'attribut (variable) il s'agit. Chaque objet stocke sa propre valeur pour chacune de ses variables.

4. Les méthodes

Une méthode est une fonction ou une transformation qui peut être appliquée aux objets ou par les objets dans une classe.

Accélérer et freiner sont des méthodes de la classe Voiture.

Tous les objets d'une même classe partagent les mêmes méthodes. Chaque méthode a un objet cible comme argument implicite (c'est l'objet lui-même : **this**, elle peut donc accéder à chacun des attributs).

La même opération peut s'appliquer à de nombreuses classes différentes.

On dit qu'elle est **polymorphe**, c'est à dire qu'elle prend différentes formes dans des classes différentes.

Une méthode peut avoir des arguments, en plus de son objet cible

> Les méthodes sont des groupes d'instructions reliées les unes aux autres au sein d'une classe d'objets qui agissent sur eux-mêmes, ainsi que sur d'autres classes et d'autres objets.
>
> Elles servent à accomplir des tâches spécifiques, au même titre que les fonctions dans d'autres langages de programmation.
>
> Elles peuvent renvoyez une valeur ou modifier l'environnement.

B)Syntaxe du langage Java

1. Explications :

1) Toutes les déclarations de classe commence par le mot réservé "**class**" suivi du nom de la classe.
La définition complète de la classe (le corps) se trouve entre accolades.

2) Dans la suite logique, nous définissons les attributs avec leurs noms précédés de leurs types.
En Java, toutes les instructions doivent être ponctuées par le **";"**. Dans cet exemple,l'attribut couleur est de type *Color*, l'attribut *calibre* est de type entier (int).
Vous remarquez qu'en Java, l'interprétation des instructions s'effectue en sens inverse

3) Cette partie montre la définition de la méthode *freiner*.
Une méthode est reconnaissable par l'utilisation obligatoire de parenthèses qui éventuellement indiquent le nombre d'arguments passés à la méthode.
Une méthode peut renvoyer une valeur.
Il faut alors indiquer son type.
Dans le cas contraire, il est impératif de préciser que la méthode ne renvoie pas de valeur et il faut placer le mot réservé "**void**". Une méthode doit toujours être définie, vous précisez alors toutes les instructions nécessaires à l'intérieur du corps de celle-ci (entre les accolades).

```
class Voiture {
   Color couleur;
   int poids;

   void accélerer() {
      ...
   }

   void freiner() {
      ...
   }
}

class Pomme {
   Color couleur;
   int calibre;
}

class Test {
   void principale () {

      Voiture mercédes, clio;

      mercédes = new Voiture();
      mercédes.couleur = "rouge";
      mercédes.accélérer();

      clio = new Voiture();
      clio.couleur = "verte";
      clio.freiner();

      Pomme pomme1 = new Pomme();
      Pomme pomme2 = new Pomme();

      pomme1.couleur = "verte";
      pomme2.calibre = 5;
   }
}
```

4) Nous ne pouvons rien écrire en dehors des classes, nous avons donc construit notre programme principal à l'intérieur d'une classe.
C'est obligatoire. Nous venons de le voir, pour exécuter des instructions, il faut passer par une méthode.

5) Déclaration des objets (variables) mercédes et clio issus de la classe Voiture (type). En java, il est possible d'utiliser les lettres accentuées.
Attention, c'est juste une déclaration, les objets ne sont pas encore créés.

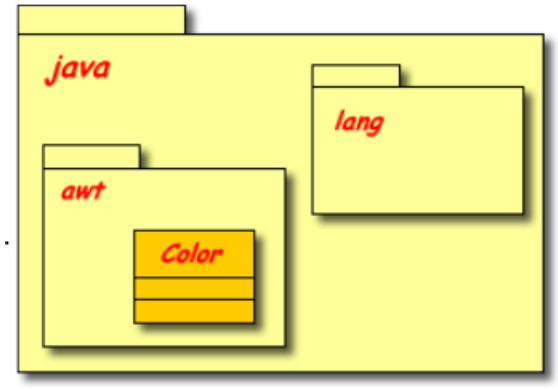
6) Création (construction) de l'objet *mercédes*. On utilise impérativement l'opérateur **new** suivi du nom de la classe (**constructeur**) suivi des parenthèses avec éventuellement à l'intérieur de ces dernières les paramètres requis. A la construction, vous avez un emplacement mémoire qui est réservé pour tout l'objet avec une capacité qui correspond à l'ensemble des attributs de la classe.

7) Une fois que l'objet est créé, il est possible d'accéder aux attributs ou au méthodes (si cela est permis) par l'utilisation du qualificateur **"."**.

8) Les objets pomme1 et pomme2 sont créés au moment de la déclaration.

C) Les Paquetages

En Java, un paquetage est une manière de regrouper des classes apparentées. Les paquetages permettent de ne rendre des groupes de classes disponibles que quand ils sont nécessaires, et ils éliminent les risques de conflits entre des noms de classes communs à des groupes de classes différents. (bien souvent, un paquetage correspond physiquement à un répertoire).



Les bibliothèques de classes Java sont contenues dans un paquetage appelé java. Les classes du paquetage javaoffrent la garantie d'être disponiblesdans toutes les implémentations de Java.

Le paquetage java contient des paquetages définissant des sous-ensembles spécifiques des fonctionnalités du langage Java, telles que les fonctions standards, le système de gestion de fichiers, les fonctions multimédia et quantité d'autres choses.

1. Utilisation d'une classe d'un paquetage

Lorsque, dans un programme, vous faites référence à une classe, le compilateur la recherche dans le paquetage par défaut (java.lang - fonctions de base du langage). Pour utiliser une classe appartenant à un autre paquetage, il est nécessaire de fournir l'information correspondante au compilateur.
Pour ce faire, vous pouvez :
- citer le nom du paquetage avec le nom de la classe,
- utiliser une instruction import en y citant soit une classe particulière d'un paquetage, soit tout un paquetage.

Citer le nom de la classe

```
class Voiture {
   java.awt.Color couleur;
   int poids;
   void accélérer() {...}
   void freiner() {...}
}

class Pomme {
   java.awt.Color couleur;
   int calibre;
}
```

Pour faire référence à une classe au sein d'un package, vous devez lister tous les paquetages contenant la classe, en les faisant suivre du nom de la classe et en séparant les différents éléments à l'aide d'un point (.).
Considérons, par exemple, la classe *Color*.
Elle est contenue dans le paquetage awt, lequel est contenu dans le paquetage java. Pour faire référence à la classe *Color* dans vos programmes, vous pouvez donc utiliser la notation ***java.awt.Color.***

En important une classe

```
import java.awt.Color;

class Voiture {
   Color couleur;
   int poids;
   void accélérer() {...}
   void freiner() {...}
}

class Pomme {
   Color couleur;
   int calibre;
}
```

L'instruction import vous permet de citer le nom (complet) d'une ou plusieurs classes, comme par exemple, import java.awt.Color;
A partir de là, vous pourrez utiliser la classe Color sans avoir à mentionner le nom de
paquetage, comme si elle appartenait au paquetage par défaut.

En important un paquetage

La démarche précédente s'avère elle aussi fastidieuse dès qu'un certain nombre de classes d'un même paquetage sont concernées.
Avec : **import java.awt.***; vous pourrez ensuite utiliser toutes les classes du paquetage awt issu du paquetage java en omettant le nom des paquetages correspondant.

D) Encapsulation

L'encapsulation est le principe qui permet de regrouper les attributs et méthodes au sein d'une classe. Cette notion est aussi associée au système de protection qui permet de contrôler la visibilité d'une variable ou d'une méthode.
En d'autres termes, cela signifie que chaque fois que vous définissez un membre d'une classe (attribut ou méthode), vous devez indiquer les droits d'accès quant à l'utilisation de ce membre.
Dans cette optique, Java propose quatre niveaux de protection :

- **public,**
- **private,**
- **protected**
- **package**

Sachez que les trois premières étiquettes de protection correspondent à des mots clés qui sont utilisés au début de la déclaration d'un attribut ou d'une méthode. Le niveau de protection package est associé à un attribut ou une méthode quand aucun des trois mots clés (**public, private** ou **protected**) n'a été indiqué.

Ce mécanisme d'encapsulation permet surtout de protéger l'objet de toute malveillance externe.
Pour cela, la plupart du temps, il faut interdire l'accès direct aux attributs et passer par les méthodes qui modifient indirectement (éventuellement) les attributs.

Par exemple : si l'on désire changer la couleur d'une voiture, cela ne se fait pas par enchantement, il est nécessaire de passer par tout un processus (décaper, poncer, passer plusieurs couches, etc...), et dans ce cas de figure l'attribut couleur ne doit pas être accessible directement.

1. Public

Tous les attributs ou méthodes d'une classe définies avec le mot clé **public** sont utilisables par tous les objets. Il s'agit du niveau le plus bas de protection. Ce type de protection est employé pour indiquer que vous pouvez utiliser sans contrainte les attributs et méthodes d'une classe.

2. Private

Tous les membres d'une classe définis avec le mot clé **private** sont utilisables uniquement par les méthodes de la classe. Cette étiquette de protection constitue le niveau le plus fort de protection. Sachez que les variables appartenant à une classe sont généralement déclarées comme privées.

3. Protected

Tous les membres d'une classe définis avec le mot clé **protected** sont utilisables uniquement par les méthodes de la classe, par les méthodes des classes dérivées et par les méthodes des classes appartenant au même package. Cette technique de protection est fortement associée à la notion d'héritage (voir ultérieurement).

4. Package

Comme indiqué plus haut, le niveau de protection **package** est employé quand aucune des trois étiquettes précédentes n'a été spécifiée. Les packages (ou paquets) permettent de

regrouper des classes Java. Cela signifie dans ce cas que tous les membres d'une classe définis avec ce niveau de protection peuvent être utilisés par les autres classes du même package.

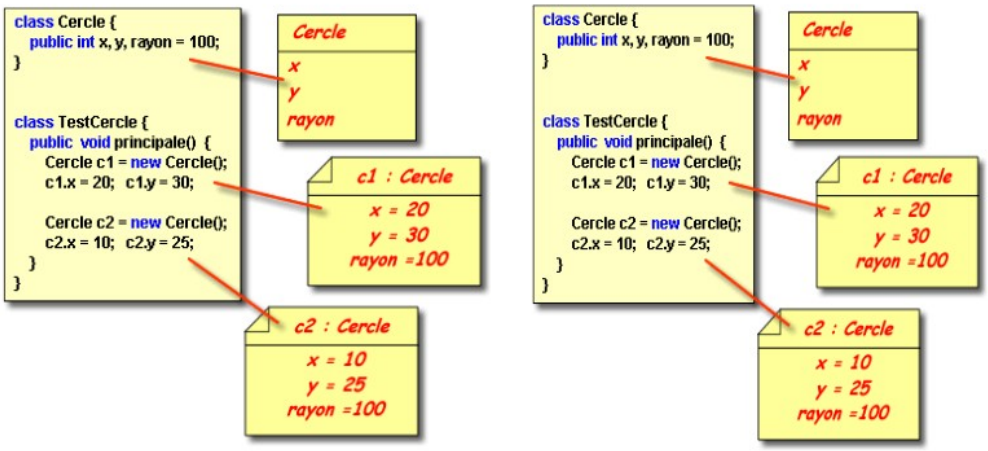
Dans l'exemple qui suit, nous avons placé des droits d'accès sur chacun des membres des classes. Ensuite dans la méthode principale de la *classe Test*, nous avons essayé d'utiliser chacun des membres des objets. Il est possible d'utiliser les membres publics ou de paquetage, par contre, les membres privés sont interdits. C'est le cas lorsque nous essayons d'accéder au membre privé couleur, il est impératif de passer par la méthode associée, à savoir : *changerCouleur*.

```
import java.awt.*;

class Voiture {
   private Color couleur;
   private int poids;
   public void accélérer() {...}
   void freiner() {...}
   public void changerCouleur(Color nouvelleCouleur) {...}
}

class Pomme {
   Color couleur;
   public int calibre;
}

class Test {
   public void principale() {
      Voiture mercedes = new Voiture();
      mercedes.couleur = "rouge";
      mercedes.changerCouleur('rouge');
      mercedes.accélérer();

      Voiture clio = new Voiture();
      clio.changerCouleur('verte');
      clio.freiner();

      Pomme pomme1 = new Pomme(), pomme2 = new Pomme();
      pomme1.couleur = 'verte';
      pomme2.calibre = 5;
}
```

E) Les membres statiques

1. Les attributs

Il semble aller de soi que la couleur d'une voiture soit une caractéristique de chacun des objets Voiture (soit par exemple mercedes un objet Voiture de couleur rouge et clio un objet Voiture de couleur verte), et non une caractéristique de sa classe. Cependant, la différence n'est pas toujours aussi évidente.
Imaginons que vous ayez besoin de créer un logiciel qui permet de tracer des cercles. Pour les besoins de votre application, les cercles auront toujours un rayon de 100 pixels. Si nous créons une classe Cercle, nous pouvons attribuer à la classe Cercle une variable rayon. Il est cependant probable que cette variable aura la valeur 100 pour toutes les instances (objets) de Cercle.
Si nous disposons d'une centaine d'objets Cercle, cela représente une grande capacité mémoire.



Il serait donc inutile que chaque instance de Cercle possède cette variable. Il suffit qu'il partage l'usage d'une variable commune à toute la classe. Ce type de variable est dit statique. Le terme correspondant en Java est **static**.

La variable rayon est aussi appelée variable de classe, il n'empêche qu'elle fait partie de c1 et de c2 et qu'il est possible d'y accéder par c1.rayon ou c2.rayon ou même par Cercle.rayon (variable de classe), mais la valeur sera toujours identique pour tous les objets de cette classe. Il est bien sûr possible de changer la valeur du rayon, par exemple par c2, et cela aura une répercussion sur l'ensemble des objets (la valeur de c1.rayon sera identique)

F) Exercice

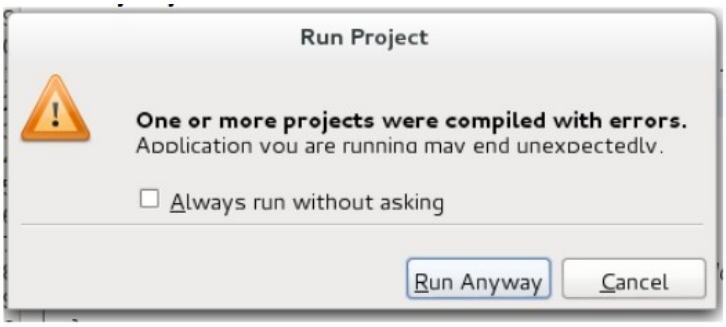
1. Exo poo1.java

```
import java.util.Date;
public class poo1{
private static int random=(int) (new Date().getTime());
public static int random(){
random=random^(random*random);
return random;
}
public static void main(String [] args){
for(;;)
System.out.println("Aleatoire : "+poo1.random());
}
}
```

Cet exemple est un générateur de nombre aléatoire. La méthode de calcul est static. Elle peut donc être appelée directement à partir de la classe, c'est d'ailleurs ce qui est fait dans le main :

```
System.out.println("Aleatoire : "+poo1.random());
```

Cette ligne se trouve dans un for(; ;) qui est donc une boucle sans fin ce qui vous conduira à la fenêtre suivante si vous lancez la compilation, cliquez sur «**Run Anyway**»



Pour arrêter l'éxecution du programme dans la **fenêtre Output** cliquez sur le symbole Stop – carré rouge)

La méthode poo1() utilise la variable static ***random***
. Cette variable, de type int, est initialisée à la déclaration. La valeur d'initialisation de random est le résultat de l'invocation de la méthode getTime() de l'objet créé à partir de la classe java.util.Date.
On peut considérer que la variable random est initialisée au nombre de millisecondes depuis le 1er janvier 1970 au moment ou le programme UNIX a été démarré.
Chaque appel à la méthode poo1 va modifier la variable random en effectuant un ou exclusif entre sa valeur et le résultat de son carré. Cela donnera à chaque exécution une valeur différente qui peut « s'apparenter » à un nombre aléatoire.

Enfin on remarque le **cast** à l'appel de getTime() :

```
private static int random=(int) (new Date().getTime());
```

En effet, getTime() est une méthode qui retourne un long. Il y aura perte d'information ce qui ne nous gène pas pour cet exemple.

2. robot.java

```
class Robot {
String status;
```

```
int speed;
float temperature;
void checkTemperature() {
if (temperature > 660) {
status = "retour au bercail";
speed = 5;
}
}
void showAttributes() {
System.out.println("Statut : " + status);
System.out.println("Vitesse : " + speed);
System.out.println("Température : " + temperature);
}
public static void main(String[] arguments) {
Robot dante = new Robot();
dante.status = "explorant";
dante.speed = 2;
dante.temperature = 510;
dante.showAttributes();
System.out.println("Augmentation vitesse ... 3.");
dante.speed = 3;
dante.showAttributes();
System.out.println("Changement température ... 670.");
dante.temperature = 670;
dante.showAttributes();
System.out.println("Vérification de la température.");
dante.checkTemperature();
dante.showAttributes();
}
}
```

- La méthode main() est créée et nommée.
- Un nouvel objet *Robot* est crée en utilisant cette classe comme modèle, le nom *dante* est attribué à cet objet
- Des valeurs sont attribuées à trois variables d'instance de l'objet *dante* : *status* est paramétrée sur le texte « Explorant », *speed* est paramétrée sur 2 et température est paramétrée sur 510.
- Sur cette ligne et plusieurs lignes qui suivent, la méthode ***showattribute***() de l'objet *dante* est appelée.
  Cette méthode affiche les valeurs actuelles des variables d'instance *status*, *speed* et *température*.
- La variable d'instance ***speed*** est paramétrée sur la valeur 3.
- La variable d'instance ***température*** est paramétrée sur la valeur 670
- La méthode ***checktemperature***() de l'objet *dante* est appelée.
  Cette méthode vérifie que la variable d'instance température est supérieure à 660.

Dans ce cas, de nouvelles valeurs sont attribuées à *status* et à *speed*.

3. Exercice 3 :

A partir de l’exemple précédent, faites en sorte de créer 2 classes une contenant les méthodes *checkTemperature* et *showAttributes* et l’autre contenant la méthode main()

Indentez votre code et commenter le !!!