TP N° 1 : Concepts de base de l'approche orientée objets- implémentation sous java

Exercice 1 : Définition, création, instanciation de classes et appel de méthodes

1- La classe voiture contient deux attributs (vitesse et nom), ainsi qu'un constructeur qui prend deux arguments du même type que les attributs, et deux méthodes « incrémentation » et « décrémentation » sans arguments qui retournent la nouvelle vitesse.

2- La classe feu de signalisation contient deux attributs (couleur et position), ainsi qu'un constructeur qui prend deux arguments, et une méthode changeCouleur qui retourne un String (la couleur). Chaque appelle à cette méthode permet de changer la couleur du feu de signalisation (par exemple du vert à l'oranger).

3- Réalisez une classe « exécuter » qui contient la méthode principale, ainsi que deux instanciations ; la première concerne un objet de type voiture « voitureDevant » avec les arguments « golf » et « 0 ».

- La deuxième instanciation concerne un objet de type feu de signalisation « feudesignalisationDevant » avec les arguments « 1» et « 13.5».
- Charger la méthode « changeCouleur » et tester si elle retourne vert ; si oui alors faite appelle à la méthode incrémentation pour incrémenter la vitesse de la voiture de 50. Sinon elle reste statique.
- Afficher les résultats dans les deux cas.

Exemples d'exécution:

Cas1 d'exécution :

```
le feu de signalisation est de couleur oranger la voiture de type golf attend encore devant le feu de signatisation
```

Cas2 d'exécution

```
le feu de signalisation est de couleur vert la voiture de type golf a une nouvelle vitesse de 50
```

Exercice 2 : Association 1..1, dépendance

Un employé travaille chez un et un seul employeur. Il est défini par son nom, une méthode qui retourne son nom et une méthode qui affiche les informations de son employeur. Un employeur est défini par un nom et le nombre de ses employés. Il contient deux méthodes : une retourne son nom et l'autre affiche ses informations (son nom et le nombre de ses employés). L'employé imprime de temps en temps des documents en utilisant une imprimante. Cette dernière, est définie par un nom et une méthode impression.

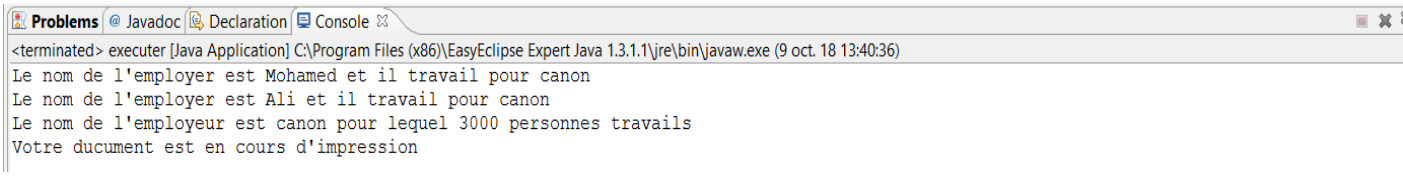
Analyse et Conception sous forme de questions !!

1. Créer un projet de travail qui contient les quatre classes : employer, employeur, imprimante et exécuter.
   - La classe employeur se compose de :
     (1) un constructeur avec deux arguments : le nom de l'employeur et le nombre de ces employés;
     (2) une méthode qui retourne le nom de l'employeur ;
     (3) une méthode qui affiche les informations d'un employeur.
   - La classe employer se compose de :
     (1) un constructeur avec deux arguments : le nom de l'employé et une variable de type employeur. Le constructeur devra afficher lors de la création d'un objet de type employer : le nom de l'employé et pour qui il travail ;

(2) une méthode qui retourne le nom de l'employé ;

- La classe imprimante se compose de :
  (1) un constructeur qui prend un argument (nom de l'imprimante) ;
  (2) une méthode qui s'appelle "impression" et qui affiche "votre document et en cours d'impression".
- Dans la classe exécuter, créez : (1) un objet de type employeur "Telecom" avec 3000 employés, (2) deux objets de type employer "Mohamed" et "Ali", (3) Affichez les informations concernant l'employeur. créez : (4) un nouvel objet de type "imprimante" (Imprimante1 qui a pour nom "Canon"), (5) Faite en sorte que l'objet Ali envoie un message à la l'imprimante pour qu'il imprime.

Exemple d'exécution :



Exercice 3 : Ensemble de valeurs de multiplicité possibles

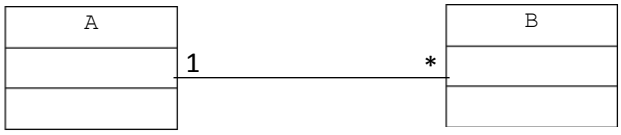
1..1 ou 1	Un et un seul
0..1	Zéro ou 1
n..m	De n à m
ou 0..	De zéro à plusieurs
1..*	De un à plusieurs

Les associations entre classes sont simplement représentées par des références. Les classes concernées possèdent en attribut une ou plusieurs références vers l'autre classe. Le nombre de référence dépend de la cardinalité. Le tableau suivant montre la correspondance entre les différentes valeurs de multiplicité possibles et le code java:

UML	JAVA

Cas d'association 1..1
 
	Lorsque la cardinalité est "1" ou "0..1", il n'y a qu'une seule référence de l'autre classe;
	`public class A {`
	`  Private B leB;`
	`  …}`
	`public class B {`
	`  private A leA;`
	`}`

Cas d'association 0..1
 
	`public class A {`
	`  Private B leB;` **//l'attribut leB est initialisé dans le constructeur.**
	`  …}`
	`public class B {`
	`  private A leA;` **//l'attribut leA n'est pas initialisé dans le constructeur car B peut ne pas avoir A.**
	`}`

Cas d'association 1..*
 
	Lorsque la cardinalité est "*", on n'aurait plus une seule référence, mais un ensemble de référence vers des objets B. soit sous la forme d'un tableau si le nombre est fixé au départ, soit sous la forme d'une collection (ArrayList), qui est un type de tableau particulier dont la taille peut varier.
	`public class A {`
	`  Private ArrayList lesB = new ArrayList();`
	`  s}`

	`public class B {`
	`  private A leA;`
	`}`

Navigabilité restreinte :
 
 	
	Pas de référence à l'objet de la classe A;
	`public class A {`
	`  Private ArrayList lesB = new ArrayList();`
	`  }`
	`public class B {`
	`  }`

Enoncé du problème:

Un étudiant est défini par nom et un prénom et une méthode toString qui permet d'afficher le nom et le prénom de l'étudiant. Un crayon est défini par un type, une couleur et deux méthodes getType et getCouleur.

1. Les classes étudiant et crayon qui sont pour le moment indépendantes. Leurs codes java équivalents sont dans le tableau suivant. La classe test contient : (1) la création d'un objet de type crayon avec affichage de ses informations, (2) la création d'un objet de type étudiant avec affichage de ses informations.

```
public class crayon {
   private String marque;
   private String couleur;
public crayon (String m, String
   c) {
  this.marque = m;
  this.couleur = c;
}
public String getMarque() {
  return this.marque;
}
public String
   afficherInformation() {
 return "Marque:"+this.marque+"
   Couleur:"+this.couleur;
}
}
```

```
public class etudiant {
   private String nom;
   private String prenom;

// Constructeur
public etudiant (String n ,
String p){
   this.nom = n;
   this.prenom = p;
}
// Afficher Information
public String
afficherInformation(){
  return "Nom:"+ this.nom+"
prenom:"+this.prenom;
}
}
```

```
class test{
public static void main(String args[]){
    crayon crayon1 = new crayon("Maped" , "Rouge");
    System.out.println(crayon1.afficherInformation());
    etudiant etudiant1 = new etudiant("BELADI", "LINA");
    System.out.println(etudiant1.afficherInformation());
}
}
```

2. Supposons qu'un étudiant possède au maximum un crayon. Sachant que lorsque l'étudiant est créé, il ne possède aucun crayon, comment faire en java ?
   (1) Comment donner un crayon à l'étudiant (il faut que le crayon existe…).
   (2) Il faut ajouter une méthode (dans étudiant) qui valorise (donne une valeur) la propriété privée **lecrayon** (référence de l'objet crayon dans étudiant) de l'étudiant.
   (3) Si l'étudiant perd son crayon, il ne possède plus de crayon. Il faut donc affecter la valeur **null** à la propriété **leCrayon** de l'étudiant (il faut donc une nouvelle méthode **perdCrayon** dans la classe étudiant).
3. Un étudiant a en général plusieurs crayons. Il faut donc attribuer une **collection** de crayons dans la classe **« étudiant ».**
   Comment déclarer et utiliser les collections (ArrayList) sous java :
   Déclaration du type ArrayList : private ArrayList mesCrayons ;

Création d'un ArrayList : this. mesCrayons = new ArrayList() ;
Ajout dans un ArrayList : this. mesCrayons.add(unCrayon) ;
Supprimer un objet d'un ArrayList : this. mesCrayons.remove(unCrayon) ;
get(int index) retourne l'élément à l'indice demandé ;
isEmpty() renvoie « vrai » si l'objet est vide ;
removeAll() efface tout le contenu de l'objet ;

Exercice 4 : Héritage, polymorphisme et abstraction

On souhaite définir un ensemble de classes pour "modéliser" des oiseaux. On veut munir chaque oiseau d'une méthode nommée décrire (méthode qui affiche le texte: 'Dans la famille des oiseaux : '). On veut créer un mélange d'oiseaux (les canaris et les perroquets) dans la classe test et appeler la méthode décrire de chacun d'eux. Les oiseaux de type Canaris ont un attribut **couleur** et une méthode **getcoleur** qui récupère la couleur d'un canari.

1. Proposer une modélisation (puis un code java) de cet énoncé en utilisant l'héritage, le polymorphisme et l'abstraction.

Exercice 5 : Agrégation

Un institue se compose de plusieurs départements. Il est défini par nom. Chaque département se compose de plusieurs étudiants. Un département est défini par un nom. Un étudiant est défini par un nom, un prénom et le nom du département dans lequel il est inscrit.

1. Modéliser puis implémenter les différentes classes en java ;
2. Créer quatre étudiants ;
3. Ajouter deux au département d'informatique et deux au département de biologie ;
4. Créer un objet de type institue (nommée INI) et lui ajouter la liste des départements ;
5. Afficher le nombre des étudiants par institue ;

Exercice 6 : Composition

Une libraire se compose d'un ensemble de livres. Elle contient une méthode qui retourne le nombre des livres. Un livre est défini par un titre et un auteur.

1. Modéliser et implémenter la relation de composition dans cet exemple.
2. Créer trois objets de type livre. Créer un objet de type libraire et lui ajoute les trois livres ;
3. Afficher les trois livres créer ;