BCOO Sujet TD0

PreReq	1. Je sais programmer, j'ai conscience qu'il faut réfléchir avant de se lancer dans le codage. J'ai des notions de structure de données.
ObjTD	Comprendre ce qu'est une **classe**.
Durée	**1 séance** de 1,5h

> *Tout objet est instance d'une certaine classe.*
>
> — *Définition*

1. Différence(s) entre objet et classe

Que pensez-vous du texte ci-dessous ?

> *Parmi les pays d'Europe, nous pouvons identifier l'Italien. L'Italien a une chaîne de montagnes qui le traverse du nord au sud et il aime la bonne cuisine, souvent à base d'huile d'olive. Son climat est de type méditerranéen, et il parle un langage admirablement musical.*
>
> — *B. Meyer*
> *Conception et Programmation Orientée Objet – 2007*

2. Différence(s) entre donnée et variable

Information vs. donnée

```
Une donnée (e.g., 37.2) est brute, elle n'a de signification que lorsqu'elle devient une
information (e.g., "température en degré Celcius").
```

Quelles sont les différences entre les concepts de :

information
variable
donnée
paramètre

3. Une simple classe

3.1. Caractéristiques

L'exemple choisi est celui de la notion de point, telle qu'elle pourrait apparaître dans un système graphique à deux dimensions :

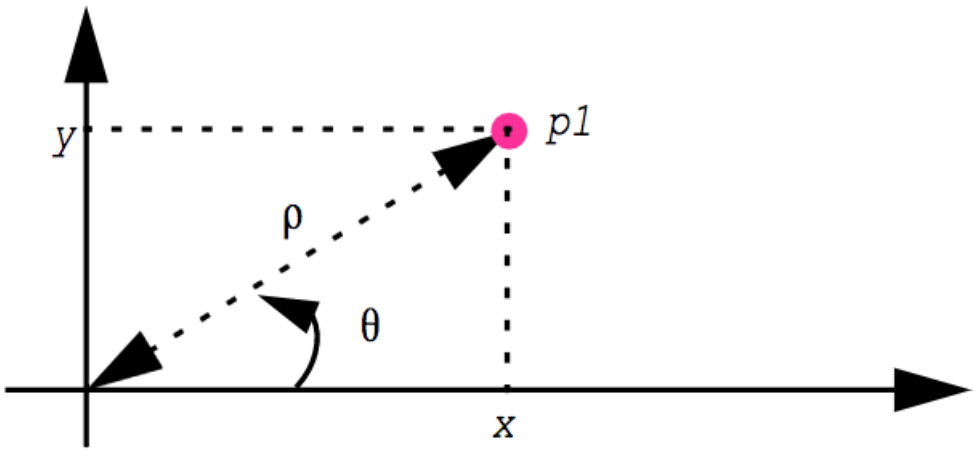

Figure 1. Source : B. Meyer, Conception et Programmation Orientée Objet — 2007

Pour caractériser le type `POINT` nous aurons besoin de quatre éléments : `x`, `y`, `rho`, `theta`.

`x` donne l'abscisse d'un point (coordonnée horizontale),

`y` son ordonnée (coordonnée verticale),

`ρ` **(rho)** sa distance à l'origine,

`θ` **(theta)** l'angle avec l'axe horizontal.

Les valeurs de `x` et `y` d'un point sont appelées ses coordonnées cartésiennes. Celles de `ρ` et `θ` ses coordonnées polaires. Une autre information utile est la `distance`, qui renvoie la distance entre deux points.

La spécification du type abstrait de donnée (ADT - *Abstract Data Type*) préciserait ensuite les commandes comme `translate` (pour déplacer un point d'une certaine distance horizontale et verticale), `rotate` (pour tourner le point d'un certain angle autour de l'origine) et `scale` (pour approcher ou éloigner le point de l'origine selon un certain facteur).

Il va falloir choisir comment fournir ces caractéristiques.

Question

Proposez une implémentation (sans les codes des méthodes) de la classe `Point`.

Décomposez les caractéristiques en fonction des aspects suivants :

```
Caractéristique -> Opération (pour les calculs)
Caractéristique -> Attribut (pour ce qui est stocké)
Opération -> Méthode (pas de résultat)
Opération -> Fonction (retourne un résultat)
```

4. Rappel sur les opérations

Un ensemble d'opérations définit le comportement de l'objet (ex : `setVitesse(valeur)`), c'est à dire son interface.

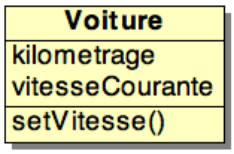

Figure 2. Exemple de classe avec attributs et opération

En objet, on parle aussi de *message* pour désigner l'accès aux caractéristiques d'un objet.

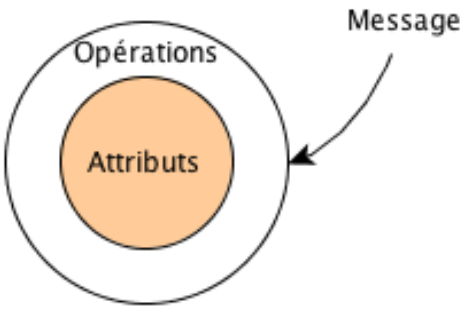

Figure 3. Principe d'encapsulation

4.1. Opérations et Visibilité

L'encapsulation

- facilite l'évolution d'une application car elle stabilise l'utilisation des objets. On peut modifier l'implémentation des attributs d'un objet sans modifier son interface
- garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'*accesseurs*). Un objet n'est manipulable qu'à travers son interface

Rappel : chaque opération a un argument implicite qui est l'objet sur lequel elle porte.
` `Int getKilometrage( );`

Exemple : `varKm = v2.getKilometrage( );`

Type d'opérations

Un accesseur `getX()` permet de consulter l'attribut `X` de l'objet, le modificateur `setX(val)` permet de modifier la valeur de l'attribut `X` avec le paramètre `val`. Par défaut, on doit avoir un accesseur par attribut privé.

4.2. Paquetages

Les paquetages permettent de regrouper les éléments de modélisation. Ils peuvent contenir d'autres sous-paquetages sans limites de niveaux.

Le paquetage est un espace de **nommage**.

Un paquetage peut importer une classe issue d'un autre paquetage.

Exemple : `Vehicules::Voitures` signifie que la classe `Voiture` est importée du paquetage `Vehicules`.

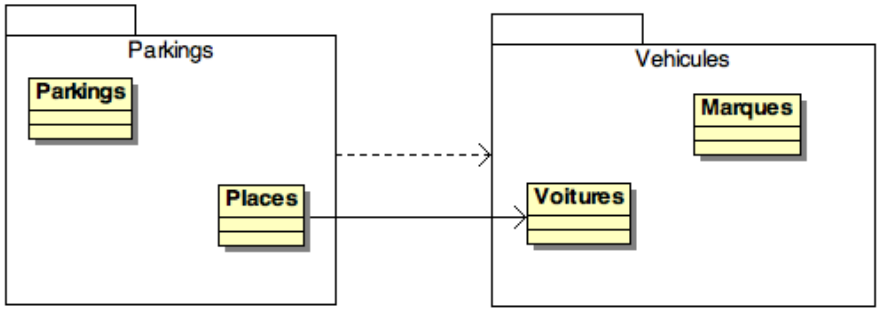

Figure 4. Dépendances entre packages

On emploiera souvent dans ce cours le terme anglais de *package* pour désigner un paquetage.

4.3. Génération de code

Voici quelques exemples de diagramme de classes et du code java associé.

4.3.1. Classe

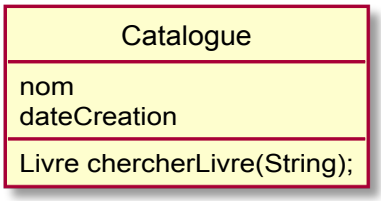

Figure 5. La classe `Catalogue`

```
import java.util.Date;

public class Catalogue {
    private String nom;
    private Date dateCreation;

    public Catalogue() {
        ...
    }

    public Livre chercherLivre(String isbn) {
        ...
    }
}
```

4.3.2. Généralisation

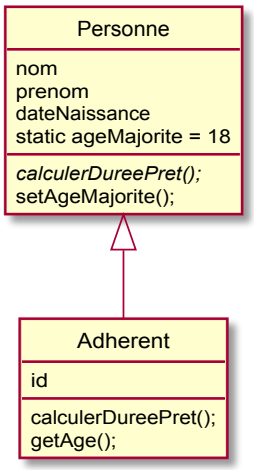

Figure 6. La classe `Adherent` hérite de `Personne`

```
public abstract class Personne {
    private String nom;
    private String prenom;
    protected Date dateNaissance;
    private static int ageMajorite = 18;
    public abstract int calculerDureePret() {... }
    public static void setAgeMajorite (int aMaj) {... }
}

public class Adherent extends Personne {
    private int id;

    public Adherent() { ... }
    public int getAge() { ... }
    public int calculerDureePret() { ... }
}
```

Si les notions de visibilité (`public`, `private`, ... ou de méthodes `abstract` ne sont pas vues ou maîtrisées, prendre le temps d'en discuter en TD.

5. Diagramme de classe

À partir de la classe `Point`, définir la classe `Ligne` qui sera composée de 2 `Point` par lesquels elle passe.

Question 1

Écrire le code Java d'une telle classe

Question 2

Dessinez le diagramme de classe correspondant

6. Démos d'outil

La suite de ce TD est destiné à vous présenter les outils (simples) qui seront utilisés ce semestre pour écrire des diagrammes de classe et faire le lien entre code et classes

6.1. plantuml

Site de l'outil

http://plantuml.com/

Démo du plugin eclipse

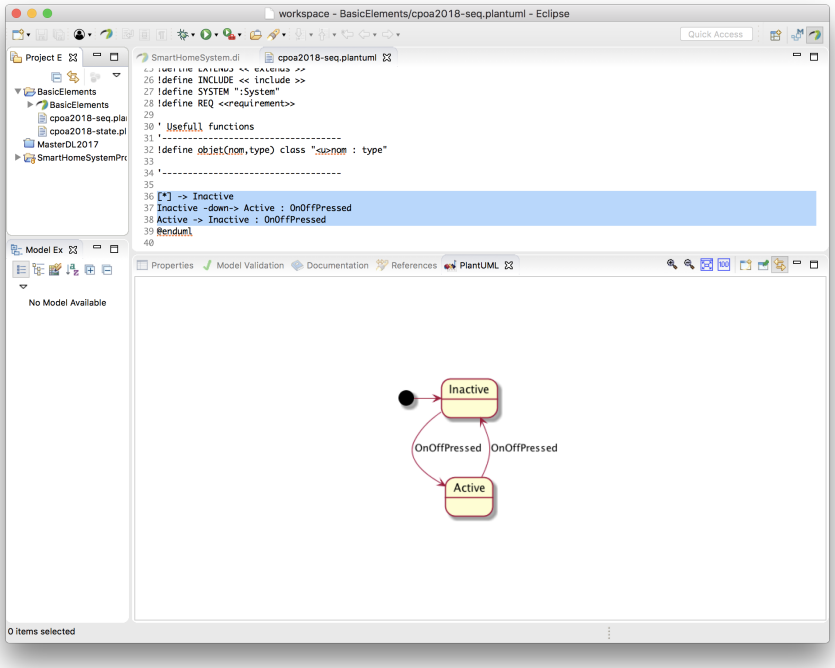

Figure 7. plantUML sous Eclipse

6.2. Umple

Site de l'outil

https://cruise.eecs.uottawa.ca/umple/

7. Pour aller plus loin...

Dans les langages objets "puristes" comme Eiffel, on applique le principe d'accès uniforme qui ne différencie pas entre attribut et méthode. `p1.x` donne la caractéristique `x` de l'objet `p1`, que ce soit par mémoire ou par calcul.