Java Mercredi

Programme:

Matin poo et exo 5,6,7

Après-midi sujet a rendre jeudi

La programmation orientée objet

La programmation orientée objet est un paradigme de programmation qui permet de représenter des objets du monde réel en utilisant des classes et des objets.

Les classes

Voici la syntaxe pour creer une classe en java :

```
public class NomDeLaClasse {
   // attributs / champs
   // constructeurs
   // methodes
}
```

Les classes sont constituées de champs, de constructeurs et de méthodes. Les champs représentent les données de la classe, les constructeurs permettent de créer des objets et les méthodes permettent de manipuler les objets.

A ces notions fondamentales vont s'ajouter trois concepts importants :

- l'encapsulation
- l'heritage
- le polymorphisme

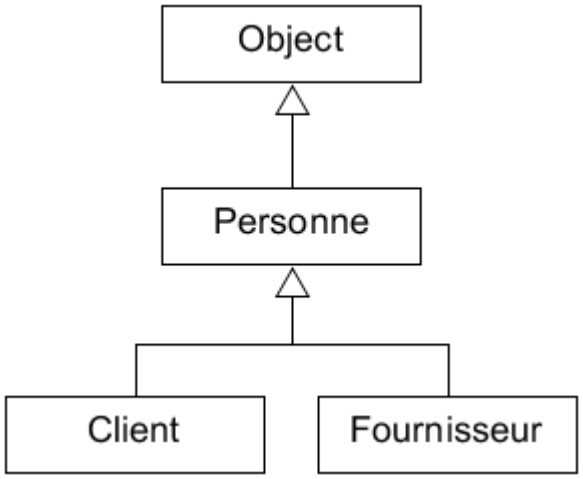
L'encapsulation permet de protéger les données d'une classe en les rendant privés. On va ainsi cacher certains éléments qui ne sont pas nécessaires à l'utilisateur de la classe, mais nécessaires à son bon fonctionnement. Dans le cas d'une classe Article, l'utilisateur n'a pas besoin de connaître le prix de revient de l'article, mais il a besoin de connaître son prix de vente. Dans le cas d'une voiture, l'utilisateur n'a pas besoin de connaître la vitesse de rotation du moteur, mais il a besoin de connaître la vitesse de la voiture.

Les éléments de la classe visibles depuis l'extérieur constituent l'interface de la classe.

L'héritage permet de creer des classes filles à partir d'une classe mère. Autrement dit, on crée une nouvelle classe à partir d'une classe existante. La classe fille hérite de tous les éléments de la classe mère. On peut alors ajouter des éléments supplémentaires à la classe fille.

Le polymorphisme permet de manipuler des objets de classes différentes avec une même interface. Ce concept est un peu difficile a comprendre au départ, mais il est très important en programmation orientée objet. Il est possible d'utiliser plusieurs classes différentes qui ont une même interface. Deux autres concepts sont souvent associés au polymorphisme : la surcharge et la substitution.

Mise en pratique avec JAVA:



Déclaration d'une classe :

```
[modificateur] class NomDeLaClasse [extends NomDeLaClasseMere] [implements NomDeLinterface  ]  {

   // Code source de la classe
   // attributs / champs
   // constructeurs
   // methodes
}
```

Les modificateurs de classes :

public : la classe est visible depuis n'importe quelle autre classe. Sans ce modificateur, la classe est visible uniquement depuis le package dans lequel elle est déclarée.

abstract: la classe est abstraite. Une classe abstraite ne peut pas être instanciée.
final: la classe ne peut pas être héritée.

Les méthodes de classes :

```
  [modificateur] typeDeRetour nomDeLaMethode([parametres]) {
      // code source de la methode
  }
```

Les modificateurs de méthodes :

public : la méthode est visible depuis n'importe quelle autre classe. Sans ce modificateur, la méthode est visible uniquement depuis le package dans lequel elle est déclarée.
protected : la méthode est visible depuis la classe, le package et les classes filles.
private : la méthode est visible uniquement depuis la classe.
Static : la méthode est une méthode de classe. Elle peut être utilisée sans instancier la classe.
abstract : la méthode est abstraite. Elle doit être redéfinie dans les classes filles.
final : la méthode ne peut pas être redéfinie dans les classes filles.
native : la méthode est implémentée dans un langage autre que Java.
synchronized : la méthode ne peut être utilisée que par un seul thread à la fois.

Création d'accesseurs pour les champs privés (getter et setter):

```
  public typeDeRetour getNomDuChamp() {
      return nomDuChamp;
  }

  public void setNomDuChamp(typeDuChamp nomDuChamp) {
      this.nomDuChamp = nomDuChamp;
  }
```

Les constructeurs d'une classe :

Il s'agit d'une méthode particulière qui permet de créer des objets à partir d'une classe. Le constructeur porte le même nom que la classe et n'a pas de type de retour. Il est possible de créer plusieurs constructeurs dans une classe. On parle de surcharge de constructeurs. Si aucun constructeur n'est déclaré dans une classe, un constructeur par défaut est créé par le compilateur. Ce constructeur par défaut est un constructeur vide qui ne fait rien.

```
  public NomDeLaClasse() {
      // code source du constructeur
  }
```

Déclaration de la classe Personne:

```
public class Personne {
   // attributs / champs
   private String nom;
   public String prenom;

   // accessible depuis la classe, le package et les classes filles
   protected String adresse;
   private String codePostal;
   private String ville;

   // constructeurs
   public Personne() {
   }
   public Personne(String nom, String prenom, String adresse, String codePostal, String ville)
{
      this.nom = nom;
      this.prenom = prenom;
      this.adresse = adresse;
      this.codePostal = codePostal;
      this.ville = ville;
   }
   // méthodes
   public String getNom() {
      return nom;
   }
   public void setNom(String nom) {
      this.nom = nom;
   }
   public String getPrenom() {
      return prenom;
   }
   public void setPrenom(String prenom) {
      this.prenom = prenom;
   }
   public String getAdresse() {
      return adresse;
   }
   public void setAdresse(String adresse) {
```

```
        this.adresse = adresse;
    }
    public String getCodePostal() {
        return codePostal;
    }
    public void setCodePostal(String codePostal) {
        this.codePostal = codePostal;
    }
    public String getVille() {
        return ville;
    }
    public void setVille(String ville) {
        this.ville = ville;
    }
    @Override
    public String toString() {
        return "Personne{" +
                "nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", adresse='" + adresse + '\'' +
                ", codePostal='" + codePostal + '\'' +
                ", ville='" + ville + '\'' +
                '}';
    }
}
```