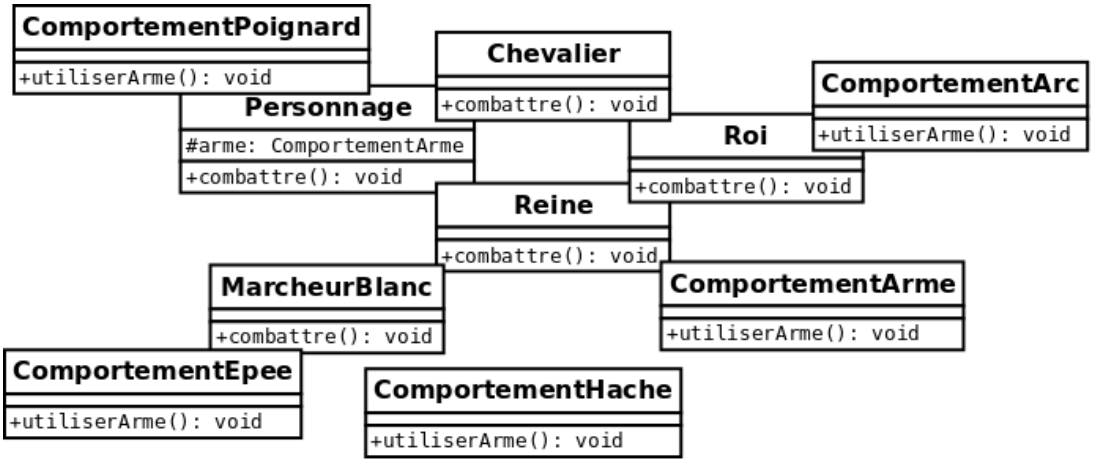
TD n°2 : Pattern Strategy

Exercice 1 (Game of Thrones)

Vous trouverez ci-dessous un ensemble de classes et d'interface pour un jeu d'aventure, toutes mélangées. Il y a des classes pour les personnages et des classes pour les comportements correspondant aux armes que les personnes peuvent utiliser. Chaque personnes ne peut faire usage que d'une arme à la fois, mais il peut en changer à tout moment en cours de jeu.

1. Identifier une classe abstraite, une interface et huit classes ordinaires.
2. Tracer les relations entre les classes (association, spécialisation, réalisation).
3. Placer la méthode setArme() dans la bonne classe.

```
void setArme(ComportementArme arme)
{
  this.arme = arme;
}
```



Exercice 2 (Intelligence artificielle d'un jeu de stratégie)

Vous devez programmer l'IA d'un jeu de stratégie en temps réel, où le but est de gérer une armée et d'aller casser la figure de l'adversaire.

Vous avez trois stratégies à votre disposition : agressive, normale et défensive. Le but de l'exercice est de pouvoir assigner et appliquer une stratégie à votre IA à chaud, c'est-à-dire dynamiquement, pendant l'exécution du programme.

1. Écrire l'interface Strategie qui ne contiendra qu'une et une seule méthode, la méthode `void action()`.

2. Écrire une classe IA qui aura comme attributs privés son nom (`String nom`) et sa stratégie courante (`Strategie strategie`). On définira un constructeur initialisant le nom et une méthode `void action()` qui affichera le nom de l'IA et qui appellera la méthode `action()` de sa stratégie courante. Enfin, IA contiendra les accesseurs à son nom et à sa stratégie.

3. Écrire les trois implémentations StrategieAgressive, StrategieNormale et StrategieDefensive de l'interface Strategie. Chaque stratégie ne consistera qu'à afficher sa nature : `Agressif`, `Normal`, ou `Défensif`.

4. Écrire ce qui sera affiché à l'écran si vous exécutez votre code avec le programme suivant :

```
public class IAMain
{
  public static void main( String[] args )
  {
    IA ia1 = new IA( "Chucky" );
    IA ia2 = new IA( "George" );
    IA ia3 = new IA( "AIUR" );

    ia1.setStrategie( new StrategieAgressive() );
    ia2.setStrategie( new StrategieDefensive() );
    ia3.setStrategie( new StrategieNormale() );

    ia1.action();
    ia2.action();
    ia3.action();

    System.out.println( "Nouvelles strategies : " +
                        "'Chucky' va fuir.\n" +
                        "'George' va passer en mode berserk.\n" );

    ia1.setStrategie( new StrategieDefensive() );
    ia2.setStrategie( new StrategieAgressive() );

    ia1.action();
    ia2.action();
    ia3.action();
  }
}
```

Exercice 3 (Statistiques et algorithmes de tri)

La classe Stat ci-dessous prend un tableau d'entiers et retourne le min, max et la médiane de ses valeurs. Rappel : la médiane d'un ensemble trié est le point permettant de séparer cet ensemble en 2 groupes de taille égale. Par exemple, pour l'ensemble `[1, 4, 6, 42, 123]`, la médiane est 6 et sépare cet ensemble en deux groupes `[1, 4]` (tout ce qui est avant 6) et `[42, 123]` (tout ce qui est après 6). Pour les ensembles de taille paire, on prend comme médiane la moyenne des 2 éléments du milieu. Ainsi pour l'ensemble `[1, 4, 42, 123]`, les deux éléments du milieu sont 4 et 42, et la médiane de l'ensemble sera 23.

Comme vous pouvez le constater, l'implémentation donnée de Stat est couplée avec l'algorithme de tri-bulle.

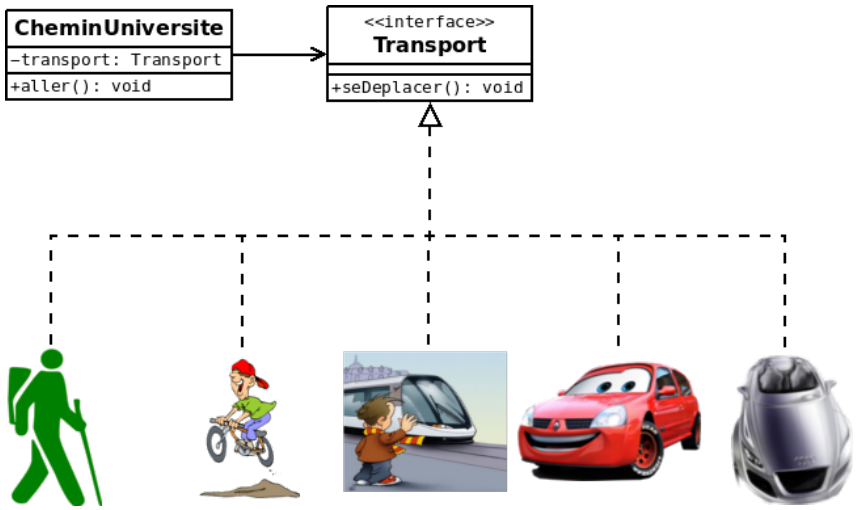
```
class Stat
{
  private int[] tableau;

  public Stat(int taille)
  {
    tableau = new int[taille];
  }

  public int getMinimum()
  {
    // instructions pour trouver le minimum
    ...
  }

  public int getMaximum()
  {
    // instructions pour trouver le maximum
    ...
  }

  public double getMedianne()
  {
    // Avant de calculer la mediane, il faut trier le tableau.

    // tri-bulle
    for( int i = tableau.length - 1; i > 0; --i )
      for( int j = 0; j < i; ++j )
        if( tableau[j] > tableau[j+1] )
          {
            int temp = tableau[j];
            tableau[j] = tableau[j+1];
            tableau[j+1] = temp;
          }

    // instructions pour calculer la mediane
    ...
  }
}
```

Or, on aimerait que Stat ne dépende pas d'un algorithme de tri fixé, mais qu'au contraire on laisse le choix à l'utilisateur (ici, un programmeur utilisant Stat) entre plusieurs algorithmes de tri. Le but ici n'est pas d'écrire ces différents algorithmes ni d'écrire les méthodes de Stat, mais de restructurer le programme de manière à ce qu'il soit facilement adaptable à plusieurs algorithmes de tri. Avant de se lancer dans du Java, c'est toujours une bonne idée de faire un diagramme de classes.

Exercice 4 (Transport)

Les étudiants ont plusieurs moyens de transport pour venir à l'université : à pied, en vélo, en tram, en Clio et en Audi TT.

1. Écrire les classes correspondant au diagramme ci-dessous.
2. Écrire un programme principal contenant 5 étudiants avec des moyens de transport différents et changeant de transport au cours du programme.



Exercice 5 (Cryptage de données)

En vous aidant du pattern Strategy, écrire les classes d'un programme cryptant des données (disons une String), avec la possibilité de choisir son système de cryptage (md5, sha256, sha512). Attention, on ne demande pas ici d'implémenter ces différents systèmes de cryptage.