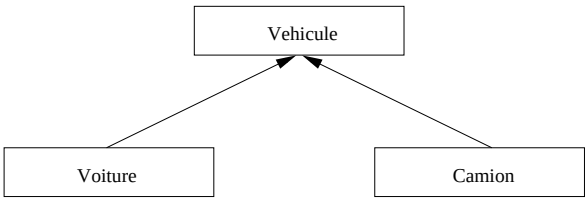
Objectifs du TD

Ce TD à pour but de présenter le concept d'héritage en Java. Il présente également un exercice sur la manipulation de tableau (via un exercice de tri).

Rappel : Héritage

Un cas concret : La classe Voiture représente toutes sortes de voitures possibles. On pourrait définir un camion comme une voiture très longue, très haute ? Mais un camion a des spécificités vis-à-vis des voitures : remorque,... On pourrait créer une classe Camion qui ressemble à la classe Voiture. Mais on ne veut pas réécrire tout ce qu'elles ont en commun.

La solution : La classe Vehicule contient tout ce qu'il y a de commun à Camion et Voiture. Camion par exemple ne contient que ce qu'il y a de spécifique aux camions.



Exercice 1

Supposons que l'on dispose de la classe `Date` suivante :

```
public class Date {
        private int jour, mois, annee;
        public Date(int j, int m, int a) {
                jour = j ; mois =m ; annee = a ;
        }
        public void setDate(int j, int m, int a) {
                jour = j ; mois =m ; annee = a ;
        }
        public String toString(){
                return jour+"/"+mois+"/"+annee ;
        }
}
```

On veut définir une classe `DateEvenement` qui associe à une date donnée un événement qui la caractérise. La solution triviale serait de définir la classe `DateEvenement` en redéfinissant cette classe en prenant exemple sur la classe `Date` et en ajoutant les nouvelles fonctionnalités que l'on juge utiles.

```
public class DateEvenement() {
        private int jour, mois, année;
```

```
        private String event = null ;

        public DateEvenement(int j, int m, int a, String e) {
                jour = j ; mois =m ; annee = a ; event = e ;
        }
        public void setDateEvenement(int j, int m, int a) {
                jour = j ; mois =m ; annee = a ; event = e ;
        }
        public String toString(){
                return jour+"/"+mois+"/"+annee+"->"+event ;
        }
}
```

Ecrire `DateEvenement` en utilisant l'héritage.

Exercice 2

1. Ecrire une classe `Animal` qui dispose d'un attribut entier `nbpattes`. Cette classe dispose des méthodes suivantes :
   – Le constructeur, qui prend en argument un entier (le nombre de pattes),
   – `String toString()` qui renvoie une chaîne de caractères contenant le nombre de pattes de l'animal,
   – `affiche()` qui affiche le nombre de pattes de l'animal.
2. Ecrire une classe `Autruche` qui hérite de `Animal`.
3. Ecrire une classe `Lapin` qui hérite de `Animal`.
4. Ecrire une classe `Main` dans laquelle la méthode `main()` crée un lapin et une autruche.

Exercice 3

Soit le programme Java suivant :

```
// fichier MainShape.java
class Shape {
    double x, y ;

    public Shape() {
        x = 0 ; y = 0 ;
    }
    public Shape(double x, double y) {
        this.x = x ; this.y = y ;
    }
    public String toString() {
        return "Position : (" + x + "," + y + ")" ;
    }
}
```

```
class Circle extends Shape {
    final static double PI = 3.141592564 ;

    private double radius ;

    public Circle() {
        radius = 0 ;
    }
    public Circle(double x, double y, double r) {
        super(x,y) ;
        radius = r ;
    }
    public String toString() {
        return super.toString() + " Rayon : " + radius ;
    }
}

public class MainShape {

    public static void main(String[] args) {
        Circle c1,c2 ;
        c1 = new Circle(1,1,3) ;
        c2 = new Circle() ;
        System.out.println(c1.toString() + "\n" + c2.toString()) ;
    }
}
```

a) De quelles variables d'instance de shape hérite la classe `Circle` ?

b) La variable `radius` étant déclarée *private*, on ne peut la modifier de l'extérieur de la classe. Ajouter une méthode `setRadius` pour pouvoir le fixer et `getRadius` pour le lire.

c) Ajoutez une variable `surface` à la classe `Circle`. Modifiez les méthodes `setRadius` et `toString`. Ajoutez les méthodes d'accès à ce champ.

Exercice 4 :

Qu'affiche le programme suivant ?

```
// fichier Main.java
class A{
    public void affiche(){
        System.out.println("Je suis un A");
    }
}
class B extends A{}
class C extends A{
    public void affiche(){
```

```
        System.out.println("Je suis un C");
    }
}
class D extends C{
    public void affiche(){
        System.out.println("Je suis un D");
    }
}
class E extends B{}
class F extends C{}

public class Main{
    public static void main(String[] args) {
        A a = new A(); a.affiche();
        B b = new B(); b.affiche();
        C c = new C(); c.affiche();
        D d = new D(); d.affiche();
        E e = new E(); e.affiche();
        F f = new F(); f.affiche();
    }
}
```

Tirez les conclusions sur l'héritage et le masquage de méthodes.

Exercice 5

On souhaite réaliser le tri d'entiers passés en paramètres de la ligne de commande. On considère un tableau d'entiers `tab`.
Une première façon de trier les entiers est de procéder par recherches successives du minimum :

1. on recherche le minimum dans le tableau entre les indices 0 et $n-1$, on l'échange avec `tab[0]` (remarque : on a besoin de l'indice du minimum dans `tab` et non du minimum lui-même),
2. on cherche alors le minimum dans le tableau entre les indices 1 et $n-1$, on l'échange avec `tab[1]`,
3. on continue ainsi de suite jusqu'à effectuer une recherche du minimum entre les indices $n-1$ et $n-1$ dans le tableau.

a) Ecrire la classe `TriTab` contenant notamment les méthodes `rechercheMin(int i, int j)` (retournant l'indice du minimum dans `tab` entre les indices i et j) et `tri` (de type void et triant `tab`) ainsi que l'attribut `private int[] tab`.
Ecrire également la classe `AppelTri` utilisant la classe précédente pour trier une liste d'entiers passés en paramètres de la ligne de commande.

b) Proposez un autre procédé de tri.