Classes abstraites et Interfaces

Qu'est ce qu'une classe abstraite

Une classe abstraite est une classe incomplète. Elle regroupe un ensemble de attributs et de méthodes mais certaines de ses méthodes ne contiennent pas d'instructions, elles devront être définies dans une classe héritant de cette classe abstraite.

A quoi ça sert ?

En général à définir les grandes lignes du comportement d'une classe d'objets sans forcer l'implémentation des détails de l'algorithme.

Pourquoi "abstraite" ?

Une classe est **abstraite** soit parce qu'on n'est pas capable d'écrire l'implémentation de toutes les méthodes, soit parce qu'on ne veut pas créer d'instance de cette classe.

Une sous-classe qui n'implémente pas toutes les méthodes abstraites de sa super-classe est elle-même abstraite. Il faut donc la qualifier d'abstraite.

Quand on ne peut pas écrire d'implémentation pour une méthode donnée, cette méthode est qualifiée d'**abstraite**. Cela signifie que l'on laisse le soin aux sous-classes d'implémenter cette méthode.

En java, c'est le mot clef **abstract** qui permet de qualifier d'abstraite une classe ou une méthode

Règles d'utilisation des classes abstraites

Règle 1 : Une classe est automatiquement abstraite si une de ses méthodes est abstraite.
Règle 2 : On peut aussi déclarer qu'une classe est abstraite par le mot clef abstract.
Règle 3 : Une classe abstraite n'est pas instanciable (on ne peut pas utiliser les constructeurs d' une classe abstraite et donc on ne peut pas créer d'objet de cette classe.).
Règle 4 : Une classe qui hérite d'une classe abstraite ne devient concrète que si elle implémente toutes les méthodes abstraites de la classe dont elle hérite.
Règle 5 : Une méthode abstraite ne contient pas de corps, mais doit être implémentée dans les sous-classes non abstraites:

abstract nomDeMéthode (<arguments>);

ce n'est qu'une signature
Règle 6 : une classe est abstraite peut contenir des méthodes non abstraites et des déclarations de variables ordinaires.
Règle 7 : Une classe abstraite peut être dérivée en une sous-classe :

abstraite si une ou plusieures méthodes ne sont pas implémentées par la classe fille,
non abstraite si toutes les méthodes abstraites sont implémentées dans la classe fille.

Le Rôle des classes abstraites :

dénomination commune de classes que l'on ne veut pas voir exister en tant que tel.
outil de spécification

Exemple

On modélise la tarification du transport d'une marchandise :

Une marchandise comporte un poids et un volume

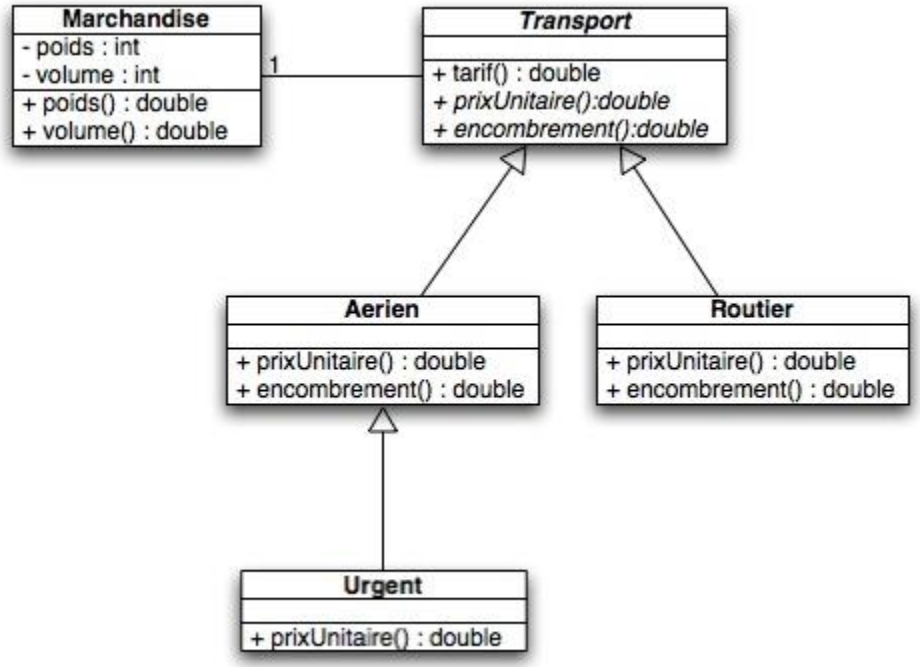
Le tarif d'un transport est toujours le produit d'un prix unitaire par l'encombrement de la marchandise transportée.

Pour le transport aérien, l'encombrement est le poids de la marchandise. Le tarif sera 10 fois supérieur à l'encombrement.

Si on veut un transport aérien urgent, il faut payer deux fois plus cher que le tarif normal.

Pour le tarif routier, l'encombrement est le volume de la marchandise. Le tarif sera 2 fois supérieur à l'encombrement.

Voici le diagramme de classes en UML



Voici les sources java (sans la classe Marchandise pour simplifier) :

```
abstract class Transport {
   protected Marchandise m ;
   public Transport (Marchandise m) {
      this.m = m ;   }
   public int tarif() {
      return prixUnitaire() * encombrement() ;  }
   abstract public int prixUnitaire() ;
   abstract public int encombrement() ;}
```

```
class Aerien extends Transport{
    public Aerien (Marchandise m) {
        super(m) ;   }
    protected int prixUnitaire(){   return 10 ;  }
    protected int encombrement(){
        return m.poids() ;  }
}
```

```
class Urgent extends Aerien{
    public Urgent (Marchandise m) {
        super(m) ;  }
    protected int prixUnitaire(){
        return 2 * super.prixUnitaire() ;  }
}
```

```
class Routier extends Transport{
    public Routier (Marchandise m){
        super(m) ;   }
    protected int prixUnitaire(){   return 2 ;  }
    protected int encombrement(){
        return m.volume() ;  }
}
```

Interfaces

A quoi sert une interface ?

Elles sont utiles pour des équipes qui travaillent sur un même projet, chacune doit pouvoir connaitre les méthodes qui seront implémentées par les autres et leur spécification pour pouvoir inclure les appels dans leurs propres méthodes.

Une interface ne comporte que des constantes et des méthodes abstraites. Elle n'est pas classe abstraite .

Une classe peut implémenter une ou plusieures interfaces, c'est à dire définir le corps de toutes les méthodes abstraites. C'est l'héritage multiple de Java.

[<modificateurs d'accès>] interface <nomInterface> [implements <interfaces>] {
<constantes> <méthodes abstraites> }

- les constantes sont explicitement ou implicitement static et final :
  - [<modificateurs>] static final <type> <nomVariable> = <valeur> ;
  - [<modificateurs>] <type> <nomVariable> = <valeur> ;
- les méthodes sont explicitement ou implicitement abstraites :
  - [<modificateur>] <type> <nomMéthode> ([<paramètres formels>]);
  - [<modificateur>] abstract <type> <nomMéthode> ([<paramètres formels>]);

Une classe qui implémente une interface doit définir le corps de toutes ses méthodes abstraites.

Les sous-classes d'une super-classe qui implémente une interface, héritent (et peuvent redéfinir) des méthodes implémentées.

Exemple d'implémentation

```
public interface Animal
    {
    // tous les animaux doivent implémenter les méthodes suivantes
    void manger(String nourriture);
    void seDeplacer();
    void respirer();
    }
```

```
public interface Parler
    {
    // une interface plus spécifique
    void parle(int phrase);
    }

  public interface Aboyer
    {
    void aboie();
    }
```

```
public class Homme implements Animal, Parler
{
//implémentations de l'interface Animal
public void manger(String nourriture)
  {
  System.out.println("Miam, " +nourriture+ "!");
  }
public void seDeplacer()
  {
  System.out.println("déplacement de l'homme");
  }
public void respirer()
  {
  System.out.println("respiration de l'homme");
  }

//implémentations de l'interface Parler
public void parle(int phrase)
  {
  System.out.println(phrase);
  }
}
```

```
public class Chien implements Animal, Aboyer
{
//implémentations de l'interface Animal
public void manger(String patee)
  {
  System.out.println("Miam, " +patee+ "!");
  }
public void seDeplacer()
  {
  System.out.println("déplacement du chien");
  }
public void respirer()
  {
  System.out.println("respiration du chien");
  }

//implémentations de l'interface Aboyer
public void aboie()
  {
  System.out.println("Ouaf!");
  }
}
```

L'héritage d'interface est aussi possible en java. A la différence des classes, l'héritage multiple est autorisé, ce qui veut dire qu'une interface peut hériter d'autant d'autres interfaces que désiré.

Exemple :

```
public interface InterfaceA {
        public void methodA();
}
public interface InterfaceB {
        public void methodB();
}
public interface InterfaceAB extends InterfaceA,
   InterfaceB {
        public void otherMethod();
}
```

```
public class ClassAB implements InterfaceAB{
	public void methodA(){
		System.out.println("A");
	}

	public void methodB(){
		System.out.println("B");
	}
	public void otherMethod(){
		System.out.println("blablah");
	}

	public static void main(String[] args) {
		ClassAB classAb = new ClassAB();
		classAb.methodA();
		classAb.methodB();
		classAb.otherMethod();
	}
}
```