Langages de programmation – TP4

Objectifs:

- Interfaces en Java

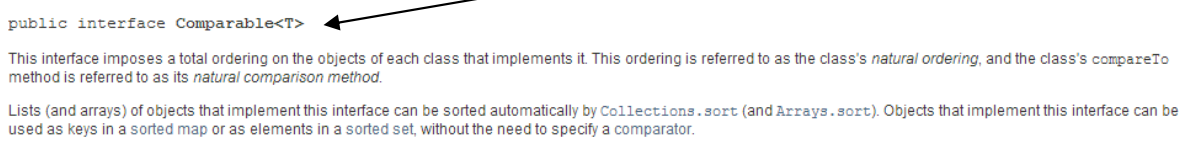
Les interfaces sont des types de données qui peuvent être définis par l'utilisateur ou peuvent être prises à partir de la bibliothèque Java, ainsi que des classes.

Une interface est une collection des comportements abstraits qui peuvent être implémentées par une classe pour ajouter un comportement qui n'est pas fourni par sa superclasse.

Après la compilation, le compilateur crée un fichier *.class* pour chaque déclaration d'interface.

Exemple:

```
public interface Taxable [extends Interface1, Interface2]
{
   int VAT=19;// sorte de constante
   int CUSTOM_TAX=10;
   public double computeVAT(); // méthodes abstraites
   public double computeRoadTax();
   public double computeCustomTax();
   public double computeTotalTax();
}
public class Vehicle implements Taxable, Comparable<Vehicle>{……..}
```



`public interface Comparable<T>`

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's `compareTo` method is referred to as its *natural comparison method*.

Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a sorted map or as elements in a sorted set, without the need to specify a comparator.

(http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html)

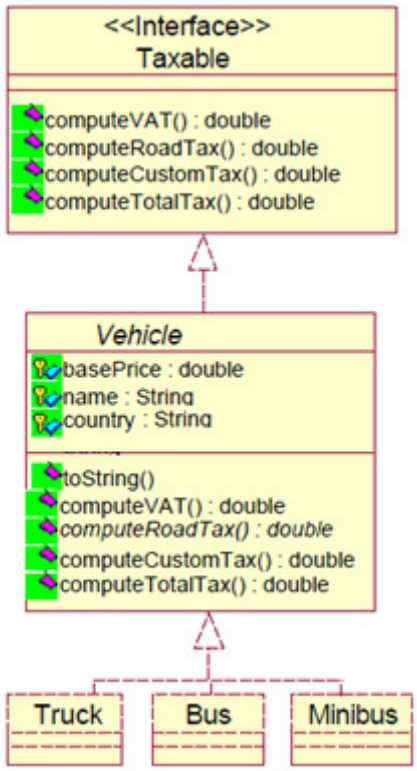
Problèmes:

Prob.1 Prenons la hiérarchie suivante des classes et l'interfaces Taxable. Les attributs de la classe Vehicle sont *protected* et les methods sont *public*.

Un concessionnaire automobile vend véhicules nationaux et étrangers: camions (*Truck*), bus et minibus. Pour calculer le prix d'un véhicule le concessionnaire doit ajouter certaines taxes demandées aux prix de base (*basePrice*) du véhicule. Les taxes pris en compte par le concessionnaire sont les suivants:

- taxe sur la valeur ajoutée (*VAT*): 19% du prix du véhicule;
- taxe de circulation (*RoadTax*): 3% pour les minibus, 4% pour les bus et 5% pour les camions;
- taxe douanière (*CustomTax*): 10% pour les véhicules fabriqués à l'étranger (country != Romania).

Afin de réutiliser le code, le modèle du système introduit la classe *Vehicle*. Il abstrait trois classes: *Truck*, *Bus* et *Minibus*, en factorisant la structure de données commune et le comportement d'eux. Le comportement du système en ce qui concerne les taxes est modélisée comme une interface avec quatre méthodes: *Taxable*. La classe *Vehicle* met en œuvre trois méthodes et offre le quatrième, *computeRoadTax (),* comme une méthode abstraite, qui sera mis en œuvre par ses sous-classes.



Tâches:

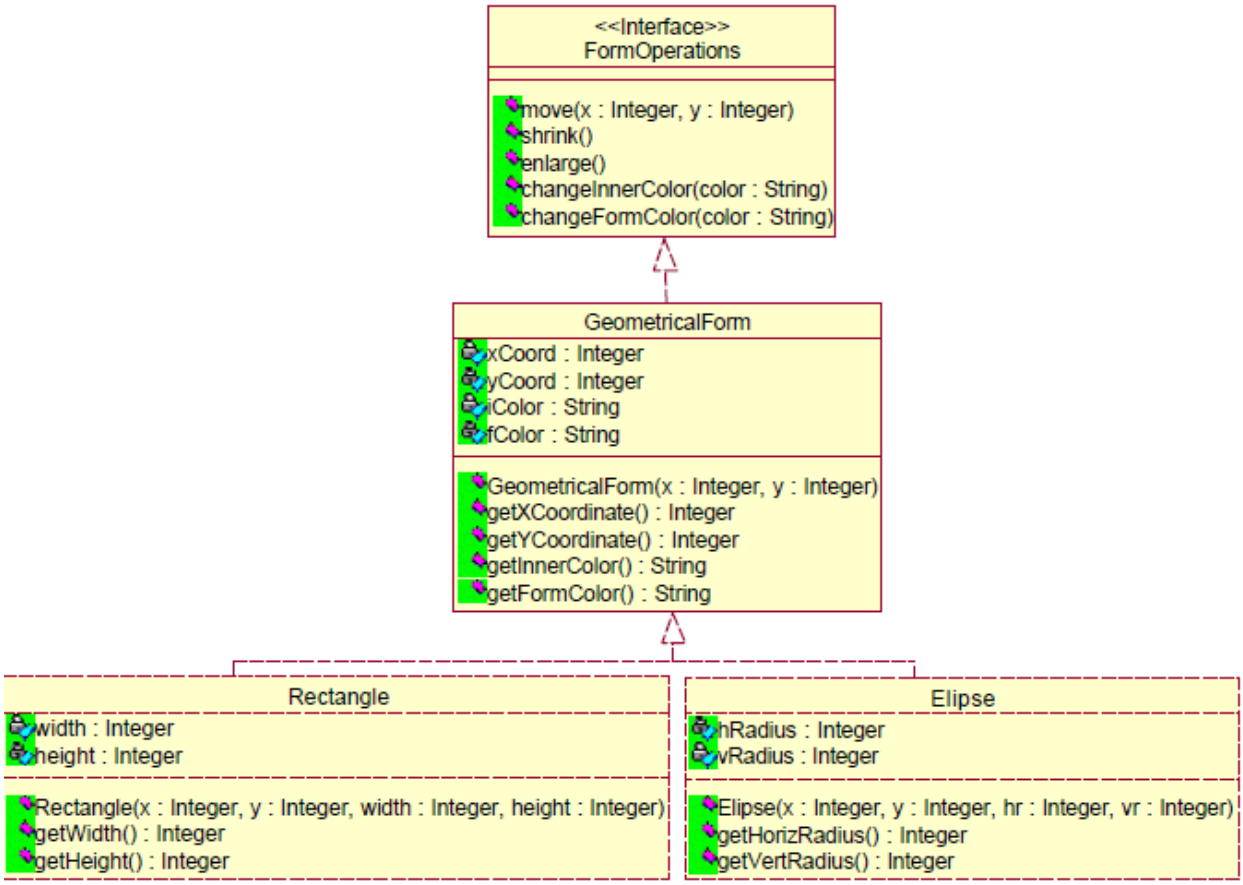
1. Mettre en œuvre le diagramme de classe ci-dessus.
2. Écrire une classe de test qui affiche pour 3 véhicules, les renseignements suivants:
   - type de véhicule: camion (Truck), bus (Bus) ou minibus (Minibus).
   - le nom de fabrication (stockées par l'attribut *name* dans la classe *Vehicle*; e.g.: Mercedes)
   - le pays de fabrication (stockées par l'attribut *country* dans la classe *Vehicle*; e.g.: Romania)
   - le prix de base en $ du véhicule (stockées par l'attribut *basePrice* dans la classe *Vehicle*; e.g.: 12000.37)
   - la somme totale des taxes qui doivent être payés.
3. Utilisez des objets créés dans la tâche 2. Triez-les après la taxe douanière (par ordre croissant). Lorsque ces valeurs sont égaux, triez les objets dans l'ordre décroissant du prix du véhicule. Utilisez l'interface Comparable, qui contient la method compareTo(o: Object).
   Exemple:

```
public int compareTo( ClassName anObj ) {
   final int BEFORE = -1;
   final int EQUAL = 0;
   final int AFTER = 1;

   if ( this == anObj ) return EQUAL;

   //primitive numbers follow this form
   if (this.classAttribute < anObj.classAttribute) return BEFORE;
   else {if (this.classAttribute > anObj.classAttribute) return AFTER;}
}
```

Prob.2. Mettez en œuvre une hiérarchie de classes entières, réelles et complexes. La partie réelle et imaginaire d'un nombre complexe sont des nombres entiers. Ecrivez l'interface NUMAR avec des operations d'addition et soustraction de deux nombres. Le résultat de chaque opération est un nouveau numéro. Les classes NUMARINTREG, NUMARREAL et NUMARCOMPLEX misent en œuvre l'interface NUMAR. Supposons que nous ne pouvons pas ajouter un nombre réel à un nombre complexe. Ecrivez une autre classe qui teste la fonctionnalité de la hiérarchie créée.

Devoir:

1. Mettez en oeuvre le diagramme de classe, puis ecrivez une test-classe avec un tableaux des objects des classes GeometricalForm, Rectangle et Ellipse et appelez les méthodes déclarées dans l'interface et les méthodes définies dans les classes.



2. Prb. 5/pp.177 ; prb. 6/pp.178