TP_TD N°3 : Héritage et polymorphisme

Exercice 1 : Héritage

On dispose de la classe suivante :

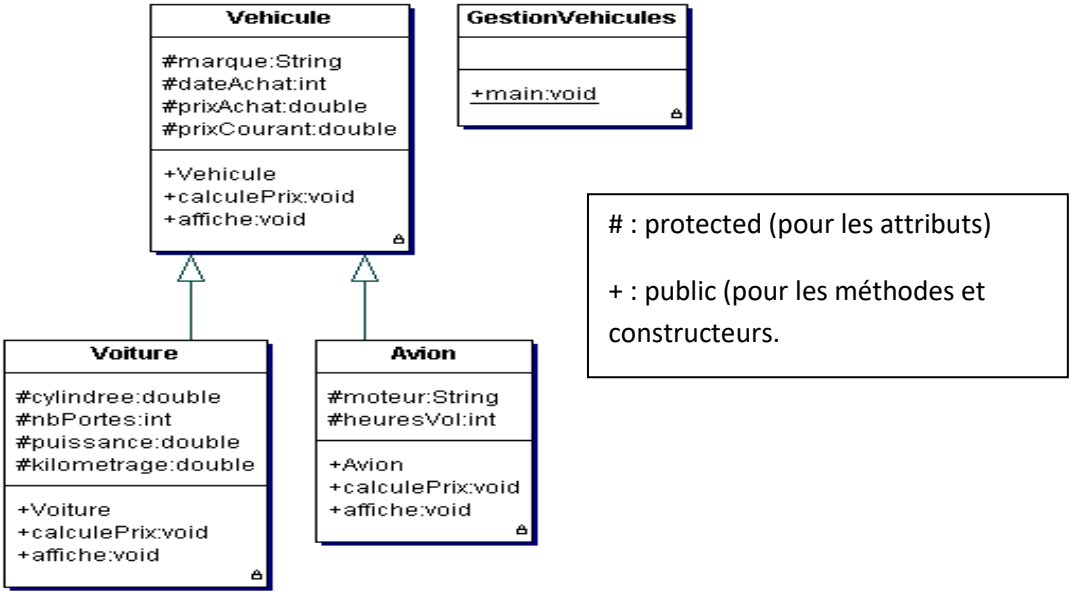
```
class Point {
protected double x,y ;
public Point(double x, double y) {
this.x=x ;
this.y=y ;
}
public void affCoord () {
System.out.println( "Coordonnées : "+ "x= "+x"+y= "+y) ; }}
```

Questions :

1. Sous Eclipse, créer un nouveau projet nommé « TP3EX1 ».
2. Créer un nouveau package nommé «Géométrie», puis créer dedans la classe Point.
3. Dedans ce package, réaliser une classe Point3D, dérivée (qui hérite) de Point permettant de manipuler des points 3D définis par leurs coordonnées(x,y,z). On y prévoira les méthodes suivantes :
   - Constructeur pour définir les coordonnées d'un objet de type Point3D.
   - affCoord() pour afficher les coordonnées d'un objet de type Point3D (sans faire appel à la méthode de la super classe affCoord()).
4. Dedans ce package toujours, réaliser une Classe PointNom3D, dérivée (qui hérite) de Point3D permettant de manipuler des points 3D définis par leurs coordonnées(x,y,z) et un nom de type caractère. On y prévoira les méthodes suivantes :
   - Constructeur pour définir les coordonnées et le nom d'un objet de type PointNom3D.
   - affCoordNom pour afficher les coordonnées et le nom d'un objet de type PointNom3D.
5. Ecrire un programme (Main) utilisant la classe PointNom3D en créant deux objets de type PointNom3D puis afficher leurs caractéristiques.
6. Exécuter le programme.
7. Changer le modificateur protected des attributs de la super classe Point3D par private.
8. Exécuter le programme à nouveau. Qu'est-ce que vous constatez…….. ?
9. Effectuer les modifications nécessaires (sans changer le modificateur private des attributs de la super classe) pour résoudre le problème rencontré dans la question précédente.

Exercice2 : Héritage et polymorphisme

Le but de cet exercice est d'implémenter les classes décrites dans le diagramme UML ci-dessous.



Questions :

1. Sous Eclipse, créer un nouveau projet nommé « TP3EX2 ».
2. Créer deux packages nommés respectivement « Transport » et « Application ».
3. Dedans le package « Transport » créer et implémenter la classes Vehicule décrite dans le diagramme ci-dessus. Définissez un constructeur prenant en paramètre les trois attributs correspondant à la marque, la date d'achat et le prix d'achat. Le prix courant sera calculé plus tard. Définissez une méthode ***public void affiche()*** qui affiche l'état de l'instance, c'est-à-dire la valeur de ses attributs. La méthode ***void calculePrix(int anneActuelle)*** fixe le prix courant au prix d'achat moins 2% par année (entre la date d'achat et la date actuelle).

4. Dedans le package « Transport » créer les deux classes Avion et Voiture qui sont décrites dans le diagramme UML mentionnés ci-dessous. Définissez, pour chacune de ces classes, un constructeur permettant l'initialisation explicite de l'ensemble des attributs, ainsi qu'une méthode affichant la valeur des attributs. Constructeurs et méthodes d'affichage devront utiliser les méthodes appropriées de la classe parente. Redéfinissez la méthode ***void calculePrix(int anneActuelle)*** dans les deux sous-classes **Voiture** et **Avion** de sorte à calculer le prix courant en fonction de certains critères, et mettre à jour l'attribut correspondant au prix courant :

   - Pour une voiture, le prix courant est égal au prix d'achat, moins :
     - 3% pour chaque année depuis l'achat jusqu'à la date actuelle
     - 5% pour chaque tranche de 10000km parcourus (on arrondit à la tranche la plus proche)

- 7% s'il s'agit d'un véhicule de marque "Renault" ou "Fiat" (ou d'autres marques de votre choix)
- et plus 10% s'il s'agit d'un véhicule de marque "Ferrari" ou "Porsche" (ou d'autres marques de votre choix).

- Pour un avion, le prix courant est égal au prix d'achat, moins :
  - 5 % pour chaque tranche de 130 heures de vol s'il s'agit d'un avion à hélices.
  - 12 % pour chaque tranche de 1100 heures de vol pour les autres types de moteurs.

Remarque : Le prix doit rester positif (donc s'il est négatif, on le met à 0)

5. Dedans le package « Application » créer et implémenter la classe exécutable GestionVehicules comme suit :

```
class GestionVehicules {

  private static int ANNEE_ACTUELLE = 2020;

  public static void main(String[] args) {

  Voiture[] garage = new Voiture[3];

  Avion[] hangar = new Avion[2];

  garage[0] = new Voiture("Peugeot", 2010, 147325.79, 2.5, 5, 180.0, 12000);

  garage[1] = new Voiture("Porsche", 2000, 250000.00, 6.5, 2, 280.0, 81320);

  garage[2] = new Voiture("Fiat", 2011, 7327.30, 1.6, 3, 65.0, 3000);

  hangar[0] = new Avion("AV123", 1998, 1730673.90, "HELICES", 1250);

  hangar[1] = new Avion("CR123", 1999, 4521098.00, "REACTION", 1500);

     for (int i = 0; i < garage.length; i++) {

        garage[i].calculePrix(ANNEE_ACTUELLE);

        garage[i].affiche(); }

    for (int i = 0; i < hangar.length; i++) {

        hangar[i].calculePrix(ANNEE_ACTUELLE);

        hangar[i].affiche();

     }}}
```

6. Améliorez la méthode **main** ci-dessus en tenant compte du fait que tous les véhicules ont un type commun: **Vehicule**.
7. Modifier le modificateur protected des attributs de la super classe Vehicule par private, puis effectuer les modifications nécessaires pour pouvoir exécuter à nouveau le programme.