Programmation Orientée Objet

TD & TP N°5
Héritage (1ère partie)

TD 5 :

Exercice 1 : Un peu d'échauffement

Soit les classes suivantes :

```
public class PointPlan{

    private float abscisse  ;
    private float ordonnee  ;

    public PointPlan( ) {  }

    public PointPlan(float x, float y){
        this.abscisse = x ;
        this.ordonnee = y ;
    }

    public PointPlan(PointPlan p){
        this(p.abscisse, p.ordonnee) ;
    }

    public float getAbscisse(){
        return this.abscisse ;
    }

    public float getOrdonnee(){
        return this.ordonnee ;
    }

    public String toString(){
        return("\n abscisse = " + this.abscisse
               + "  ordonnee = " + this.ordonnee);
    }

    public void init(){
        //lecture des valeurs de : abscisse et ordonnee
    }
} // fin classe PointPlan


public class Point3D extends PointPlan{
    private float azimut;
```

```
    public Point3D( ){
        super() ;
    }

    public Point3D(float x, float y, float z){
        this.abscisse = x;
        this.ordonnee = y;
        this.azimut = z;
    }

    public Point3D(Point3D p){
        super(p) ;
        this.azimut = p.azimut;
    }

    public float getAbscisse(){
        return this.abscisse ;
    }

    public float getOrdonnee(){
        return this.ordonnee ;
    }

    public float getAzimut(){
        return this.azimut ;
    }
}// fin classe Point3D
```

Énumérer toutes les anomalies de conception de la classe `Point3D` ainsi que les instructions rejetées par le compilateur. Justifier chaque réponse.

Exercice 2 : Communications téléphoniques

Soit la classe suivante gérant une communication téléphonique locale :

```
public class Communication{
    private float duree;   // durée de la communication
    private String numero; // numéro appelé
    public static final float PRIX_SECONDE = 0.01; // prix de la seconde

    public Communication( ) { } //constructeur vide

    // initialise une nouvelle communication de durée d et de numéro num
    public Communication(float d, String num){
        this.duree  = d;
        this.numero = num;
    }

    // constructeur par copie
    public Communication(Communication c){
        this(c.duree, c.numero) ;
    }
```

```
    // retourne la durée de la communication
    public float getDuree(){
        return this.duree ;
    }

    // retourne le numéro appelé
    public String getNumero(){
        return this.numero ;
    }

    // retourne la chaîne représentant la communication courante
    public String toString(){
           return ("\n duree = " + this.duree
           + "  numero appele = " + this.numero);
    }

    // initialise interactivement la communication courante
    public void init(){
        //lecture des valeurs de duree et numero
    }

    // retourne le prix de la communication courante
    public float prixCommunication(){
        return this.duree * Communication.PRIX_SECONDE ;
    }
} // fin classe Communication
```

On souhaite étendre cette classe de façon à gérer une communication internationale qui possède 2 données supplémentaires : le pays appelé (représenté par une chaîne de caractères) et le prix de la seconde de communication pour ce pays.

Écrire la classe `CommunicationInter` (et une classe `TestCommunicationInter`) en respectant les étapes suivantes :

1. Écrire les variables d'instances.
2. Écrire 3 constructeurs (un constructeur vide, un constructeur donnant tous les arguments nécessaires pour initialiser une communication internationale et un constructeur par copie).
3. Écrire les méthodes d'accès.
4. Si nécessaire compléter la classe de façon à pouvoir afficher, initialiser interactivement et calculer le prix d'une communication internationale.

Compléter la classe `TestCommunicationInter` au fur et à mesure des questions précédentes de façon à tester chaque constructeur et chaque méthode.

TP 5 :

— Finir le TP N°4
— Écrire et tester les classes de l'exercice 3

Exercice 3 : Histoires de western

On souhaite réaliser une application Java permettant d'écrire facilement des histoires de western. Dans nos histoires, nous auront des brigands, des cowboys, des shérifs, des barmen et des dames en détresse.

Nous allons notamment définir : - la classe de base, - étendre une classe en ajoutant des attributs, - étendre une classe en ajoutant des méthodes, - voir l'intérêt de l'instruction `super`, etc.

$\mathcal{Q}_1$: **Tous humains**

Les intervenants de nos histoires sont tous des humains. Un humain est caractérisé par son nom et sa boisson favorite. La boisson favorite d'un humain est, par défaut, de l'eau. Un humain pourra parler. On aura donc une méthode `parle(String texte)` qui affiche :

```
(nom de l'humain) - texte
```

Un humain pourra également se présenter (il dit bonjour, son nom, et indique sa boisson favorite), et boire (il dira : « `Ah ! un bon verre de [sa boisson favorite] ! GLOUPS !` »).

Réalisez la classe `Humain`.

$\mathcal{Q}_2$: **Brigand, cowboys et dames en détresses**

Les dames, les cowboys et les brigands sont tous des humains (si, si !). Par la même, ils ont tous un nom et peuvent tous se présenter. Par contre, il y a certaines différences entre ces trois types d'individus. Les brigands peuvent kidnapper les dames, les dames peuvent se faire enlever et les cowboys peuvent les libérer.

Une dame est caractérisée par la couleur de sa robe, et par son état (libre ou captive). Elle peut se faire kidnapper (auquel cas elle hurle), se faire libérer par un cowboy (elle remercie alors le héros qui l'a libérée). Elle peut également changer de robe (tout en s'écriant « `Regardez ma nouvelle robe [couleur de la robe] !` ».

Un brigand est un humain qui est caractérisé par un look (« `méchant` » par défaut), un nombre de dames enlevées, la récompense offerte lorsqu'il est capturé (100$ par défaut), un booléen indiquant s'il est en prison ou pas. Il peut kidnapper une dame (auquel cas, il s'exclame « `Ah ah ! [nom de la dame], tu es mienne désormais !` »).

Un cowboy est un humain qui est caractérisé par sa popularité (0 pour commencer, augmente à chaque dame délivrée) et un adjectif le caractérisant (« `vaillant` » par défaut). Un cowboy peut tirer sur un brigand (un commentaire indique alors : « `Le [adjectif] [nom] tire sur [nom du méchant]. PAN !` » et le cowboy s'exclame « `Prend ça, rascal !` »). Il peut également libérer une dame.

Réalisez les trois classes `Brigand`, `Cowboy` et `Dame`.

$\mathcal{Q}_3$: **Redéfinition**

Nous allons maintenant redéfinir les méthodes pour adapter les comportements de nos personnages à leur type. Quand on demande son nom a une dame, elle répond « `Miss [son nom]` » et un brigand dira « `[son nom] le [son look]` » (par exemple : « `Bob le méchant`»). Un cowboy dira simplement son nom (ce sont des gens simples).

Redéfinissez la méthode `getNom()` dans les classes `Brigand` et `Dame`.

$\mathcal{Q}_4$: **Utilisation de `super`**

On désire aussi changer le mode de présentation des brigands, dames et cowboys. Un brigand parlera aussi de son look et du nombre de dames qu'il a enlevées et de la récompense offerte pour sa capture. Une dame ne pourra s'empêcher de parler de la couleur de sa robe, alors qu'un cowboy dira ce que les autres disent de lui (son adjectif) et parlera de sa popularité.

Si on récrit la méthode qui permet a une personne de se présenter (comme à la question précédente), la méthode de la classe `Humain` sera remplacée. On désire quand même appeler cette méthode.

Le brigand Bob se présentera par exemple de la manière suivante :

```
(Bob) - Bonjour, je suis Bob le méchant et j'aime le Tord-Boyaux.
(Bob) - J'ai l'air méchant et j'ai déjà kidnappé 5 dames !
(Bob) - Ma tête est mise a prix 100$ !
```

Réalisez les modifications nécessaires pour obtenir ce résultat.

De même, on veut donner une boisson par défaut à chaque sous-classe d'humain (lait pour la dame, tord-boyaux pour le brigand et whisky pour le cowboy).

Modifiez le constructeur des différentes classes pour tenir compte de ces nouvelles exigences.