UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence INFO
Info0201

Travaux dirigés n° 5

Composition et agrégation

Exercice 1 (Ordinateurs et processeurs)
Nous considérons les classes ci-dessous.

```
public class Ordinateur {

  private Processeur processeur;

  public Ordinateur(Processeur processeur) {
    this.processeur = processeur;
  }

  public Processeur getProcesseur() {
    return processeur;
  }

  public void changerProcesseur(
      Processeur processeur) {
    this.processeur = processeur;
  }

  public String toString() {
    return "Ordi. [" +
           processeur.toString() + "]";
  }
}
```

```
public enum Fabricant {
    AMD,
    Intel,
    Samsung,
    NVIDIA
}
```

```
public class Processeur {

  private double frequence;
  private Fabricant fabricant;

  public Processeur(double frequence,
                    Fabricant fabricant) {
    this.frequence = frequence;
    this.fabricant = fabricant;
  }

  public double getFrequence() {
    return frequence;
  }

  public Fabricant getFabricant() {
    return fabricant;
  }

  public void overclocker() {
    frequence *= 1.25;
  }

  public String toString() {
    return "Fab. " + fabricant +
           " (" + frequence + "GHz)";
  }
}
```

```
public class TestOrdinateur {

  public static void main(String[] args) {
    Processeur p1 = new Processeur(2.66,
                                   Fabricant.Intel);
    Processeur p2 = new Processeur(3.2,
                                   Fabricant.AMD);
    Ordinateur ordiBob = new Ordinateur(p1);
    Ordinateur ordiRoger = new Ordinateur(p2);

    System.out.println(ordiBob.toString());
    System.out.println(ordiRoger.toString());

    // Point d'observation 1

    Ordinateur ordiRaoul = new Ordinateur(p1);
    ordiRaoul.getProcesseur().overclocker();

    System.out.println(ordiBob.toString());
    System.out.println(ordiRoger.toString());
    System.out.println(ordiRaoul.toString());

    // Point d'observation 2
  }
}
```

1°) Représentez le diagramme de classes complet.

2°) Donnez l'état mémoire complet aux deux points d'observation du `main`.

3°) Quel est l'affichage obtenu ? Est-ce normal ?

4°) Utilisez la copie profonde dans la classe `Ordinateur`. Spécifiez tous les changements à effectuer.

5°) Si l'on souhaite qu'une référence ne *pointe* vers aucun objet, il faut lui affecter la valeur `null`. Que se passe-t-il si nous passons `null` en paramètre aux différents constructeurs et méthodes de ces classes ? Proposez les modifications éventuelles.

Exercice 2 (Segments)
Pour cet exercice, nous considérons que nous disposons d'une classe `Point` représentant un point cartésien. Cette classe possède des constructeurs par défaut, par initialisation et par copie. Elle implémente l'interface `IPoint` suivante :

```
public interface IPoint {

  public double getAbscisse();

  public void setAbscisse(double abscisse);

  public double getOrdonnee();

  public void setOrdonnee(double ordonnee);

  public void translater(double dx, double dy);

}
```

Nous souhaitons écrire une classe `Segment` à partir de l'interface `IPoint` : un `Segment` est constitué de deux `IPoint` représentant ses deux extrémités.

1°) D'après vous, s'agit-il d'une composition ou d'une agrégation ? Est-ce que cela change quelque chose au niveau du code de la classe `Segment` ?

2°) Dans le cas d'une copie profonde, quel problème cela pose d'utiliser des `IPoint` ?

3°) Écrivez la classe `Segment` qui doit contenir :
- Les constructeurs par initialisation et par copie ;
- Des *getters* et *setters* pour ses extrémités ;
- Des méthodes `toString` et `afficher` ;
- Une méthode `translater` similaire à celle présente dans l'interface `IPoint`ù.

Nb : vous devez réfléchir à copier (ou non) les objets dont les références sont retournées ou passées en paramètre.

4°) Donnez le diagramme UML des classes/interfaces citées dans cet exercice.

Exercice 3 (Bibliothèque)
Nous souhaitons modéliser une bibliothèque, avec ses rayonnages et ses livres. La description suivante vous précise l'ensemble des classes ou énumérations qui sont mises en jeu, les noms des classes et des énumérations étant précisés en **gras**.

Un **Auteur** de **Livre** est caractérisé par son nom, son prénom et sa **Date** de naissance. Une **Date** correspond à un jour, un mois et une année. Un **Livre** possède un titre, un **Genre** (roman, bande dessinée ou jeunesse), un **Auteur**, une **Date** de parution et un numéro ISBN (une chaîne de caractères ou un entier long). Dans la **Bibliotheque**, nous trouvons des **Rayonnage** (par défaut, aucun) dans lesquels les **Livre** sont rangés. Pour simplifier, chaque **Rayonnage** possède un nombre de places qui est spécifié lors de sa création (et qui ne peut être modifié). Une place correspond à un **Livre** (ou `null` si aucun livre n'est présent à cette place). Les places sont numérotées de 0 à *n - 1* où *n* correspond au nombre de places maximum. Les **Rayonnage** d'une **Bibliotheque** sont aussi identifiés par un numéro de 0 à *n - 1* où *n* correspond au nombre de **Rayonnage** dans la **Bibliotheque**.

Pour un **Auteur**, une **Date** ou un **Livre**, il est possible de récupérer l'ensemble des informations, sans pouvoir les modifier. Ces informations seront donc précisées au moment de l'instanciation. Un **Rayonnage** possède une méthode `getLivre` qui retourne un **Livre** situé à une position spécifiée en paramètre. La méthode `setLivre` permet de placer un **Livre** à une position donnée dans un **Rayonnage**, le **Livre** et la position sont spécifiés en paramètre. Il est possible d'ajouter un nouveau **Rayonnage** dans une **Bibliotheque** à l'aide de la méthode `ajouterRayonnage`. Une méthode `getRayonnage` permet de récupérer le `Rayonnage` d'une **Bibliotheque** dont le numéro est spécifié en paramètre. Toutes les classes possèdent une méthode `toString`.

Donnez le diagramme UML complet répondant à cette description.