Conception et programmation avancées

TD révisions UML - JAVA

I) Partie UML

1) Les instances décrivent-elles des méthodes ? Où sont décrites les méthodes ?

Non bien sûr. Les méthodes sont décrites dans la classe et éventuelles super-classes d'une instance. Remarque : question pour vérifier que les étudiants ne pensent qu'il y ait des méthodes dans les instances. C'est l'occasion de reparler des méthodes décrites dans les super-classes et non redéfinies dans les sous-classes MAIS héritées.

2) Quelle est la différence entre une opération et une méthode ? Dans quelle situation cette différence est-elle notable ?

Une opération est conceptuelle. Une méthode est l'implémentation d'une opération. Ainsi, dans un diagramme de classes, dans Figure apparaîtra l'opération superficie() mais elle n'apparaîtra pas dans ses sous-classes car elle est héritée. Par conséquent toute sous-classe l'aura. Par contre, à l'implémentation, cette opération aura plusieurs implémentations, donc sera implémentée par plusieurs méthodes, certainement au moins une méthode par figure géométrique « concrète » : triangle, cercle, rectangle ... La différence est donc notable en cas de polymorphisme et donc de redéfinition/surcharge car une opération aura plusieurs implémentations et donc plusieurs méthodes. En conception, on dit que toute figure saura calculer sa superficie. En implémentation, on dit comment chaque figure le fera.

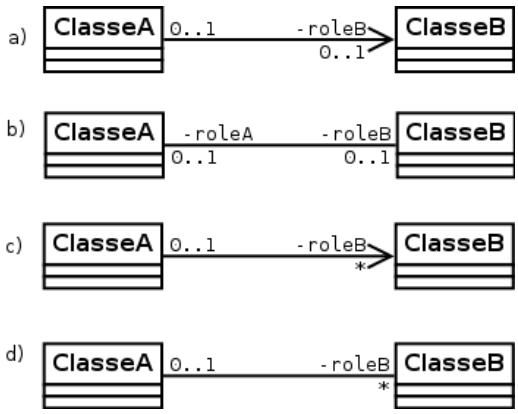
3) Citer un diagramme par type de diagrammes

a. Diagrammes statiques : Diagramme de classes ou diagramme d'objets.

b. Diagrammes dynamiques : Diagramme de cas d'utilisation

c Diagrammes d'interactions : Diagramme de séquences

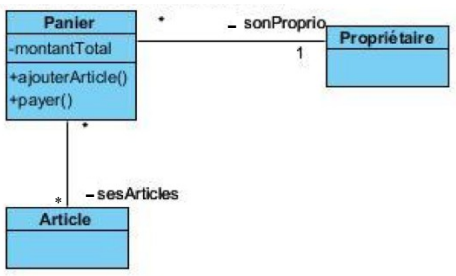
4) Proposez une implémentation java pour chacune des situations suivantes :



```
a) public classA{
       private ClasseB roleB ;
       ….
}
public classB{
       // la classeB ne connaît pas l'existence de classA !!!
}

b)  public classA{
       private ClasseB roleB ;
       ….
}
public classB{
        private ClasseA roleA ;
}

c) public classA{
       private ArrayList<ClasseB> roleB ;
       ….
}
public classB{
       // la classeB ne connaît pas l'existence de classA !!!
}

d) public classA{
       private ArrayList<ClasseB> roleB ;
}
public classB{
       private ClasseA roleA ;
}
```

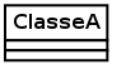
5) Faites de la rétro-ingénierie : proposez le diagramme de classes de ce bout de code :

```
public class Panier { private Proprietaire sonProprio ;
                      private ArrayList<Article> sesArticles;
                      private float montantTotal;
                      public Panier(Proprietaire p){...}
                      public void ajouterArticle(){...}

                      public void payer(){...}

... }
```

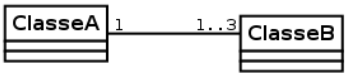


6) Proposez les plus petits diagrammes d'objets respectant les diagrammes de classe suivant et :

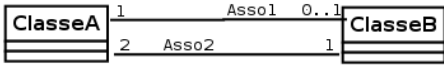
a) composé d'au moins 2 objets :



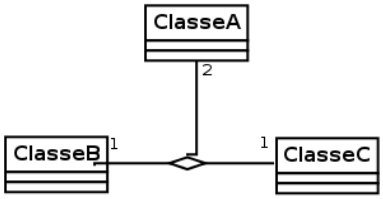
b) composé d'au moins 2 objets de ClassA :



c) composé d'au moins 1 objet :



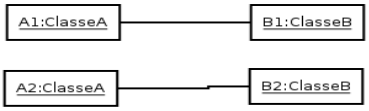
d) composé d'au moins 1 objet



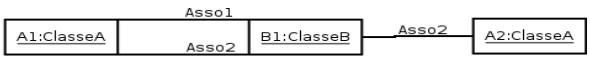
Correction

a) composé d'au moins 2 objets :

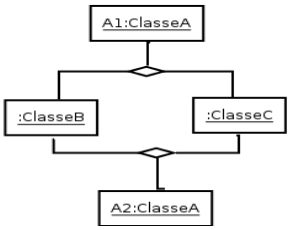
b) composé d'au moins 2 objets de ClassA :



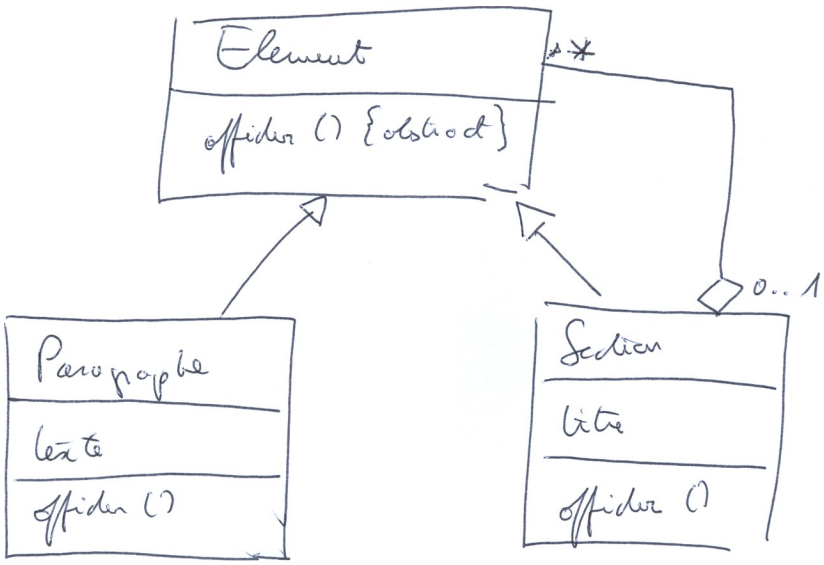
c) composé d'au moins 1 objet :



d) composé d'au moins 1 objet



7) On cherche à modéliser la structure de documents hiérarchisés. De tels documents contiennent deux types d'éléments : les paragraphes et les sections. Les sections peuvent contenir des paragraphes ainsi que des sous-sections, elles-même structurées de la même manière.
On doit pouvoir afficher tous les types d'éléments, et en ajouter aux sections et sous sections.



II) Partie JAVA

1) Héritage et constructeur

L'exercice consiste seulement à deviner la sortie du programme ci-dessous (la méthode `main` de la classe `EssaiConstructeurs`)

```
class A {
    A() {
        System.out.println("bonjour de A");
    }
}

class B extends A {
    boolean verite;
    int valeur;

    B() {
        System.out.println("constructeur B()");
    }

    B(int valeur) {
        this();
        this.valeur = valeur;
        System.out.println("constructeur B(int)");
    }

   B(boolean verite) {
        this.verite = verite;
        System.out.println("constructeur B(boolean)");
    }

    B(boolean verite, int valeur) {
        this(valeur);
        this.verite = verite;
        System.out.println("constructeur B(boolean, int)");
    }

    public String toString() {
        return "B : (" + verite + ", " + valeur +")\n";
    }
}

class EssaiConstructeurs {
    public static void main(String[] argv) {
        B b = new B(true);
        System.out.println(b);
        b = new B(false, 5);
        System.out.println(b);
    }
}
```

Correction :
bonjour de A
constructeur B(boolean)
B : (true, 0)

bonjour de A
constructeur B()
constructeur B(int)
constructeur B(boolean, int)
B : (false, 5)

2) Classe abstraite

On définit quatre classes.

- La classe `Animal` est abstraite et déclare uniquement une méthode abstraite nommée action, sans paramètre et qui ne retourne rien.
- La classe `Chien` hérite de `Animal` et définit la méthode `action` qui écrit à l'écran "J'aboie".
- La classe `Chat` hérite de `Animal` et définit la méthode `action` qui écrit à l'écran "Je miaule".
- La classe `EssaiChat` contient trois champs statiques :
  - un champ statique pour un attribut de type `java.util.Random` qui est initialisé dès sa définition
  - une méthode statique nommée `tirage` sans paramètre qui retourne un `Animal` qui a une chance sur deux d'être un `Chat` et une chance sur deux d'être un `Chien`.
  - une méthode `main` qui utilise la méthode `tirage` et invoque la méthode `action` sur l'animal obtenu par cette méthode.

```
public abstract class Animal{
    abstract void action();
}

class Chien extends Animal{
    void action(){
        System.out.println("J'aboie");
    }
}

class Chat extends Animal{
    void action(){
        System.out.println("Je miaule");
    }
}

class EssaiAnimal{
    static java.util.Random alea = new java.util.Random();
    static Animal tirage(){
        return (Math.abs(alea.nextInt()) % 2 == 0 ? new Chien() : new Chat());
    }

    public static void main(String[] arg){
        tirage().action();
    }
}
```

3) Interface

Soit l'interface Dessinable et les classes représentant un rectangle, un cercle et un triangle. Ecrivez une classe TestDessin qui remplit un tableau contenant un triangle, deux rectangles et un cercle puis dessine chacun de ces objets.

```
interface Dessinable{
    public void dessiner();
}


class Rectangle implements Dessinable{
    public void dessiner(){
        System.out.println("je dessine un rectangle");
    }
}

class Cercle implements Dessinable{
```

```
    public void dessiner(){
        System.out.println("je dessine un cercle");
    }
}

class Triangle implements Dessinable{
    public void dessiner(){
        System.out.println("je dessine un triangle");
    }

}
```

Correction :

```
public class Test{
       public static void main(String[] args)
       {
         Dessinable[] schema = {new Triangle(), new Rectangle(),
                                new Rectangle(), new Cercle()} ;
         for(int i =0; i< 3; i++){
            schema[i].dessiner();
         }
       }
}
```