**UEF4.3. Programmation Orientée Objet**

# Corrigé de l'examen final

**Questions (6 pts)**

1. Expliquez le concept de polymorphisme et ses différents types.

Réponse: (0,5+0,5+0,5+0,5+0,25+0,25) **(2,5)**

- Le terme polymorphisme décrit la caractéristique d'un élément qui peut se présenter sous différentes formes. En POO, cela peut s'appliquer aussi bien aux objets qu'aux méthodes.
- Le polymorphisme d'objets permet une certaine flexibilité dans la manipulation des objets en exploitant la relation d'héritage entre les classes. Pour cela, il applique la règle suivante : Si la classe A est dérivée de la classe B alors un objet de la classe A peut aussi être considéré comme étant un objet de la classe B.
- Polymorphisme de méthodes exprime le fait que :
  - des méthodes d'une même classe puissent avoir le même nom et des signatures différentes (surcharge ou surdéfinition de méthodes)
  - La même méthode puisse se comporter différemment sur différentes classes de la hiérarchie. Autrement dit, des méthodes peuvent avoir des noms similaires dans une hiérarchie de classes et être différentes. Ceci est appelé redéfinition ou spécialisation de méthodes.

2. Quelle est la différence entre un programme correcte et un programme robuste ? Quel est le moyen le plus efficace et le plus "propre" pour rendre un programme java plus robuste ?

Réponse: (0,5+0,5+0,5) **(1,5)**

- Un programme est **correct** s'il accomplit bien la tâche pour laquelle il a été conçu.
- Un programme est **robuste** s'il est capable de fonctionner même en présence d'erreurs.
- On peut rendre un programme plus robuste d'une manière claire et propre grâce au mécanisme de gestion des exceptions

3. En java, comment compiler et exécuter un programme en mode console ? donner un exemple**(1)**

réponse (0,25+0,25+0,25+0,25)

- Pour compiler ce fichier, il faut utiliser la commande *javac* suivie du nom du fichier.java.
  Exemple : `javac test.java`
- Pour exécuter un programme Java, il faut utiliser la commande *java* suivie du nom du fichier. Exemple : `java test (absence du .java)`

4. Expliquez chacun des mots dans la ligne suivante **(1)**

```
public static void main(String [] args)
```

réponse (0,25+0,25+0,25+0,25)

- Le mot clé *static* précise que la méthode main est une méthode de classe, elle n'est donc liée à aucune instance (objet) particulière de la classe dans laquelle elle se trouve. En outre, comme elle porte le nom main, il s'agit de la méthode principale.

- La paramètre *String args []* de la méthode main est un tableau d'objets de type String qui permet de récupérer des arguments transmis au programme au moment de son lancement
- Le mot clé public est obligatoire pour que le programme puisse s'exécuter. Il permet à la machine virtuelle d'accéder à la méthode main.

**Exercice (14 pts)**

**Ceci est un corrigé type. Cette solution n'est pas unique.**

**Le barème est sur 68 + 2 pts pour la lisibilité du code.**

**1. classe Voisin (14,75)**

| 14.75 | Code java |
|---|---|
| 0.25 | `class Voisin {` |
| 0.25 | `private Sommet sommet;` |
| 0.25 | `private int poids;` |
| 1 | `public Voisin(Sommet s, int p){`<br>`sommet = s;`<br>`Poids = p; }` |
| 0.75 | `public Sommet getSommet(){`<br>`return sommet; }` |
| 0.75 | `public int getPoids(){`<br>`return poids; }` |
| 5,75 | `public boolean equals (Object o){`<br>`return`<br>`sommet.getEtiquette().equals(((Voisin)o).getSommet().getEtiquette()`<br>`);}` |
| 5.75 | `public int hashCode(){`<br>`return sommet.getEtiquette().hashCode();}`<br>`}`<br>`}` |

**2.classe Sommet (14.25)**

| 14.25 | Code java |
|---|---|
| 0.25 | `class Sommet {` |
| 0.25 | `private String etiquette;` |
| 1.5 | `private Set<Voisin> voisins;` |
| 2.25 | `public Sommet (String e){`<br>`etiquette = e;`<br>`voisins = new HashSet<Voisin>();}` |
| 0.75 | `public String getEtiquette(){`<br>`return etiquette; }` |

| | |
|---|---|
| 2 | Public boolean estVoisin(Sommet s){<br>boolean trouve = false;<br>boolean trouve = false;<br>Iterator<Voisin> it = voisins.iterator();<br>while(it.hasNext()){<br>if(it.next().getSommet().getEtiquette() == s.getEtiquette())<br>{trouve = true;break; }<br>}<br>return trouve;} |
| 1.25 | public void ajouterVoisin(Sommet s, int p){<br>voisins.add(new Voisin(s,p));<br>s.getVoisins()).add(new Voisin(this,p); // optionnelle} |
| 4 | public void afficherVoisins(){<br>Iterator<Voisin> it = voisins.iterator();<br>System.out.println("les voisins de " + etiquette + "sont: ");<br>while(it.hasNext()){<br>System.out.println(it.next().getSommet().getEtiquette());<br>}<br>}<br>} |
| 2 | public int degre(){<br>int degre = 0;<br>Iterator<Voisin> it = voisins.iterator();<br>while(it.hasNext()){<br>degre = degre + (it.next().getPoids());}<br>return degre;<br>}<br>} |

**2.classe Graphe (16)**

| 16.5 | Code java |
|---|---|
| 0.25 | class Graphe { |
| 3.25 | private Map <String,Sommet> sommets = new HashMap<String, Sommet>(); |
| 1.5 | public void ajouterSommet(String e){<br>sommets.put(e,new Sommet(e));} |
| 1.25 | public Sommet chercherSommet(String etiquette){<br>return sommets.get(etiquette);} |

**6.75**

```
 public void ajouterArete(Sommet a, Sommet b, int p)throws
SommetException{
        Sommet sa = chercherSommet(a.getEtiquette());
        Sommet sb = chercherSommet(b.getEtiquette());

            if(sa == null||sb == null) throw new SommetException();
            else{
                  sa.ajouterVoisin(sb,p);
                  sb.ajouterVoisin(sa,p);

/* nécessaire si instruction optionnelle de la méthode ajouterVoisin
n'a pas été prévue*/
}
}
```

**3**

```
public void afficherVoisinsSommets(){
for (String key : sommets.keySet()) {
 Sommet value = sommets.get(key);
             value.afficherVoisins();
}
}
}
```

**Class SommetException (0.25+1+1 = 3.25)**

```
public class SommetException extends Exception{}
```

**A. Class TestGraphe (5.25)**

```
class TestGraphe{
public static void main(String[] args){

      Graphe g = new Graphe();

      g.ajouterSommet(new Sommet("v1"));
      g.ajouterSommet(new Sommet("v2"););
      g.ajouterSommet(new Sommet("v3"););
      g.ajouterSommet(new Sommet("v4")););

try{  g.ajouterArete(v1, v2, 3 );
      g.ajouterArete(v1, v3, 1 );
      g.ajouterArete(v2, v3, 1 );
      g.ajouterArete(v2, v4, 2 );
      g.ajouterArete(v3, v4, 3 );

}catch (SommetException e){
            System.out.println("Sommet inexistant");
      }

      g.afficherVoisinsSommets();

}

}
```

**B.** Il suffit de remplacer HashMap par TreeMap dans la classe Graphe. **(2 pt)**

```
Map <String,Sommet> sommets = new TreeMap<String, Sommet>();
```

**C.** Il faut remplacer HashSet par TreeSet dans la classe Sommet. **(2)**

```
    public Sommet (String e){
          etiquette = e;
          voisins = new TreeSet<Voisin>();
    }
```

- La classe Voisin doit implémenter Comparable et redéfinir CompareTo. La redéfinition des méthodes equals et hashCode n'est plus nécessaire **( 10.5)**

```
class Voisin implements Comparable<Voisin>{
                ...
    ...
    ...

    public int compareTo(Voisin other){ (
    if(this.sommet.getEtiquette()==other.getSommet().getEtiquette())
    return 0;
    else
     if(this.getPoids() == other.getPoids())
    return
    sommet.getEtiquette().compareTo(other.getSommet().getEtiquette());
    else
    return this.poids - other.poids;
    }

}
```