EISTI ING1 - EXAMEN DE PROGRAMMATION JAVA – 1ERE SESSION 2014

Mercredi 4 juin 2014, durée 2H, documents non autorisés sauf une feuille A4 recto-verso.

Exercice 1. Héritage/Implémentation. [1 pt]

1.1. Une classe peut-elle hériter de plusieurs classes ?

1.2. Une classe peut-elle implémenter plusieurs interfaces ?

1.3. Une interface peut-elle hériter de plusieurs interfaces ?

Exercice 2. Mot clé final. [1 pt]

2.1. Quelle est son utilité devant une méthode ?

2.2. Quelle est son utilité devant un attribut ?

Exercice 3. Un composant simple. [4,5 pts]

On considère la classe Composant dont on vous donne le schéma UML et le code Java correspondant


```
public class Composant {
    String libellé ;
    Composant(String l) {this.libellé = l;}
}
```

3.1. Visibilités : quels sont les mots-clés Java qui traduisent les 4 visibilités UML : -, ~, # et +.

3.2. Encapsulation : que faut-il modifier dans le code de cette classe pour suivre le principe d'encapsulation ? On considère ce principe assuré dans les questions suivantes.

3.3. Contraintes sur le libellé :

3.3.1. Que faut-il mettre en place en Java pour rendre le libellé obligatoire et modifiable ?

3.3.2. Que faut-il modifier par rapport à a) pour qu'il soit non modifiable ?

3.3.3. Que faut-il modifier par rapport à a) pour qu'il soit facultatif ?

3.3.4. Est-il possible qu'il soit à la fois facultatif et non modifiable ?

3.3.5. On ajoute une contrainte de longueur minimale de libellé (3 caractères) : où doit-on prendre en compte cette contrainte ?

3.4. Egalité : on définit l'égalité de 2 composants comme l'égalité de leur libellé (même taille et mêmes caractères).

3.4.1. Donner la signature annotée de la méthode d'égalité (redéfinition de la méthode de java.lang.Object) ;

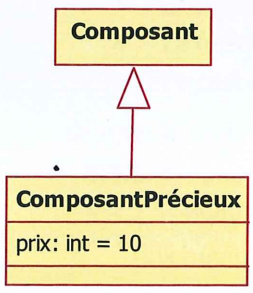
3.4.2. Donner l'instruction d'égalité entre le libellé du composant courant et celui du paramètre vu à travers la référence comp de type Composant :

3.4.2.1. si le champ libellé est obligatoire ;

3.4.2.2. si le champ libellé est facultatif.

3.4.3. Quelle autre méthode de la classe Object faut-il redéfinir ?

Exercice 4. Un composant précieux. [3,5 pts]



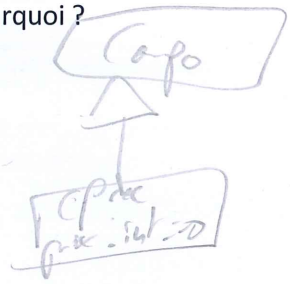
ok/ 9pts 2

On considère la classe ComposantPrecieux qui hérite de la classe Composant. Dans la classe Composant, on a un seul constructeur à un paramètre, éventuellement nul (cf exercice 3).

4.1. Est-il possible de n'écrire aucun constructeur pour la classe ComposantPrécieux ? Pourquoi ?

4.2. Est-il possible d'avoir ensemble les 4 constructeurs de signatures suivantes ?

- ComposantPrécieux() // libellé nul, prix par défaut
- ComposantPrécieux(int prix) // libellé nul
- ComposantPrécieux(String libellé) // prix par défaut
- ComposantPrécieux(String libellé, int prix)



4.3. Si non pourquoi ? Si oui, donner le code des 4 constructeurs.

4.4. La méthode toString de Composant retourne la chaîne "composant "+ libellé et celle de ComposantPrécieux "composant "+ libellé + " à " + prix + " euros".

4.4.1. Modifier la chaîne retournée dans la classe ComposantPrécieux pour réutiliser le code de la classe Composant.

4.4.2. Modifier la chaîne retournée dans la classe Composant pour obtenir "composant anonyme" si le libellé est nul (en une seule instruction)

4.4.3. Qu'affiche le code suivant ?

```
Composant  c1 = new Composant("joli") ;
ComposantPrécieux c2 = new ComposantPrécieux("très joli", 100);
Composant c3 = c2;
System.out.println(c1) ;
System.out.println(c2) ;
System.out.println(c3) ;
```

4.5. Une classe abstraite a-t-elle forcément une méthode abstraite ?

4.6. Une classe abstraite peut-elle avoir un constructeur ?

4.7. Une classe concrète hérite d'une classe abstraite contenant une méthode abstraite. Comment peut-on faire pour ne pas implémenter la méthode abstraite dans la classe concrète ?

Exercice 5. Exception. [3 pts]

5.1. Quelle est la différence entre throw et throws en Java ?

5.2. Peut-on faire un try sans catch ? Justifier.

5.3. On vous donne le code suivant d'une méthode permettant de récupérer un objet Personne dans un fichier objet par dé-sérialisation. Le nom du fichier est passé en paramètre.

```
public static Personne chercherPersonneByNuméro(String filename) {
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename));
    Personne  p = (Personne) ois.readObject();
    ois.close();
    return p;
}
```

- La construction du flux peut lever une IOException.
- La lecture peut lever une IOException ou une ClassNotFoundException.
- La fermeture du flux peut lever l'exception IOException.
- Le cast peut lever une ClassCastException (seule exception de type RuntimeException).

Donner le code modifié de cette méthode dans les 2 situations suivantes :

- **5.3.1.** propager les exceptions en cas de problème tout en s'assurant de ne pas laisser un flux ouvert ;

5.3.2. renvoyer une référence nulle en cas de problème quelconque, tout en s'assurant de ne pas laisser un flux ouvert.

Exercice 6. Collections et énumérations [3 pts]

6.1. A quel type de collection Java correspondent la cardinalité * et la contrainte {ordered} en UML ?

6.2. A quel type de collection Java correspondent la cardinalité * et la contrainte {seq} en UML ?

6.3. Quelle méthode est utilisée pour ranger ou trouver les éléments dans un HashSet ?

6.4. Quelles méthodes sont utilisées pour ranger ou trouver les éléments dans un TreeSet ?

6.5. Donner la déclaration d'une liste d'associations permettant d'associer à des priorités des files d'attente de personnes. Une priorité est représentée par un entier et une personne par la classe Personne.

6.6. Soient l'enum Jour et la classe Evénement suivantes :

```
public enum {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE}

public class Evénement {
    private Jour jour ;
    private short debut ;
    private short durée ;
    private String libellé;
    // reste de la classe non représenté : tout constructeur, getter ou setter nécessaire est supposé disponible
}
```

6.6.1. Comment peut-on parcourir les jours de la semaine ?

6.6.2. Compléter la méthode suivante qui retire tous les événements d'un ensemble correspondant à un jour particulier ? On ne vous demande pas de gérer les problèmes éventuels (ensemble non modifiable ou paramètres nuls).

```
public static void enleverEvénementParJour(Set<Evénement> événements, Jour jour) { //TODO }
```

Exercice 7. Programmation générique [4 pts]

On vous donne la javadoc et la signature de la méthode mediane ajoutée dans une classe Utils. On rappelle que la médiane d'une collection est le nombre qui sépare la collection ordonnée en valeurs croissantes en deux groupes de même effectif.

```
/**
 * Calcul de la médiane d'une collection de valeur suivant un ordre donné
 * @param <E> type des éléments
 * @param coll collection d'éléments où s'effectue la recherche
 * @param cmp comparateur définissant un ordre pour les éléments de la collection coll
 * @return la médiane des éléments de la collection coll ordonnés suivant le comparateur cmp
 * @throws IllegalArgumentException si un paramètre est null ou si la collection est vide
 */
public static <E> E mediane(Collection<E> coll, Comparator<E> cmp) throws IllegalArgumentException
```

7.1. Ecrire le code de la méthode en réutilisant au maximum les possibilités des collections de java.

7.2. La clause throws IllegalArgumentException est-elle obligatoire ? Pourquoi ? En cas de réponse négative, expliquer son intérêt.

7.3. Que faut-il changer dans la signature de la méthode pour rendre cette dernière la plus générique possible vis-à-vis de l'héritage ?

7.4. Ecrire un comparateur sur les entiers tel que :

a. les nombres pairs soient inférieurs aux nombres impairs ;
b. les nombres pairs soient dans l'ordre croissant naturel ;
c. les nombres impairs soient dans l'ordre décroissant naturel.

7.5. En réutilisant les questions précédentes, donner un extrait de code qui permet de calculer la médiane des valeurs 1, 2, 3, 4, 5, 6 et 7. Quel est le résultat ?