NFA032 Programmation objet avec Java

CNAM-centre d'enseignement de Paris

Deuxième session 2015 semestre 1

Documents et calculatrice interdits. Le barème est donné à titre indicatif.

Exercice 1 : références (3 points)

```
public class Obj{
    int[] lop;
    Obj(int x){
        lop=new int[2];
        lop[0]=x;
        lop[1]=x+1;
    }
}
public class Objet{
    public static void main(String[] args){
        Obj[] tab = new Obj[3];
        for (int i=0; i<3; i++){
            if (i!=2)
                tab[i]=new Obj(i);
        }
    }
}
```

Question 1

Représentez au moyen d'un petit dessin les objets et tableaux existant à la fin de l'exécution du programme donné ci-dessus. Chaque objet sera représenté par un rectangle et chaque référence par une flèche ou une adresse. Attention : on ne vous demande pas un diagramme de classe. C'est chaque objet créé, chaque tableau et chaque référence qui doit être représenté.

Question 2

Donnez le code java permettant d'afficher à l'écran le contenu d'une case d'un tableau appelé `lop` (n'importe quelle case de n'importe quel tableau portant ce nom).

Exercice 2 : programmation objet (5 points)

Question 1

Un logiciel comporte un nom, un numéro de version et un prix. Ecrivez un classe permettant de le représenter, avec un constructeur et une méthode d'affichage. Si vous déclarez les variables *private* (ce qui n'est pas obligatoire), écrivez aussi les méthodes permettant d'accéder à leurs valeurs.

Question 2

Une licence d'un logiciel comprend un logiciel (représenté par un objet de la classe écrite à la question 1), une année de validité (pour simplifier, on considère que la licence est valide pendant une année civile), et un numéro d'identification. Ecrivez la classe des licences avec un constructeur et une méthode d'affichage.

Question 3

Ecrivez une mathode *main* qui crée un objet instance de la classe des licences et appelle la méthode d'affichage sur cet objet.

Exercice 3 : exceptions (2,5 points)

```
public class Excep7 {
   public static void meth3(int x){
      if (x==0)
           throw new PresquePas();
      else if (x>2) {
           throw new PasDuTout();
      }
      Terminal.ecrireStringln("Fin␣normale␣meth3");
   }
   public static void meth2(String s){
       if (s==null){
           throw new PresquePas();
       }
       try {
           meth3(s.length());
           Terminal.ecrireStringln("Fin␣normale␣meth2");
       } catch (PresquePas e) {
           Terminal.ecrireStringln("PresquePas␣dans␣meth2");
      }
   }
   public static void meth1(char c, String s){
     try {
         if (c == 'o')
             meth2(s+c);
         else
             meth2(s);
         Terminal.ecrireStringln("Fin␣normale␣meth1");
     } catch (PasDuTout e ){
```

```
            Terminal.ecrireStringln("PasDuTout dans meth1");
        }
    }

    public static void main(String[] args) {
        try{
            meth1('o', "v");
            Terminal.ecrireStringln("Fin normale main premier try");
        }catch(PasDuTout e){
            Terminal.ecrireStringln("PasDuTout dans main");
        }catch(PresquePas e){
            Terminal.ecrireStringln("PresquePas dans main");
        }
        try{
            meth1('n',"oui");
            Terminal.ecrireStringln("Fin normale main second try");
        }catch(PasDuTout e){
            Terminal.ecrireStringln("PasDuTout dans main");
        }catch(PresquePas e){
            Terminal.ecrireStringln("PresquePas dans main");
        }
        Terminal.ecrireStringln("Fin normale main");
    }
}
class PasDuTout extends Error{}
class PresquePas extends Error{}
```

Donnez les messages affichés par l'exécution de ce programme.

Exercice 4 : interfaces et héritage (5,5 points)

On définit les interfaces suivantes :

```
public interface StringIterateur{
  boolean enResteTIl();
  String suivante() throws YEnAPlus;
  void reset();
}
public interface StringSDD{
  void add(String s);
  int size();
  void remove();
}
```

L'interface `StringIterateur` permet de parcourir les éléments d'une structure de données contenant des String. La méthode `suivante` permet d'obtenir l'élément suivant. Elle lève une exception s'il n'y a pas d'élément suivant. La méthode `reset` permet de recommencer au début, au premier élément. La méthode `enResteTIl` permet de tester s'il reste encore des éléments à examiner dans le pourcours de la structure. L'interface StringSDD permet d'ajouter et d'enlever des éléments d'une structure contenant plusieurs String.

1. écrivez une classe appelée `TabStringFin` qui implémente les deux interfaces, qui stocke des String dans un tableau de String. Cette classe doit ajouter et enlever les éléments à la fin du tableau.
2. écrivez une méthode qui recherche une String dans n'importe quel objet qui implémente l'interface `StringIterateur`. Cette méthode doit être dans un classe appelée `Recherche`.
3. écrivez une classe appelée `TabStringAjFinRetDeb` qui implémente les deux interfaces et qui stocke les String dans un tableau, et qui ajoute les éléments en fin de tableau. Jusqu'ici, c'est identique à la classe de la question 1, mais la différence est que l'élément enlevé par la méthode `remove` doit être le premier du tableau. Vous utiliserez l'héritage si possible.
4. Ecrivez encore une classe appelée Main avec une méthode main qui stocke une instance de `TabStringFin` et une instance de `TabStringAjFinRetDeb` dans un tableau à deux cases, puis qui prend au clavier une chaîne de caractère et détermine si elle appartient à l'un des deux objets.

Exercice 5 : structures récursives (4 points)

```
class Militaire{
  String nom;
  String grade;
  Militaire superieur;
  Militaire(String n, String g, Militaire s){
    nom=n;
    grade=g;
    superieur=s;
  }
}
```

1. Créez dans une méthode main des instances de la classe Militaire contenant un matelot appelé Popeye, un quartier-maître appelé Riri et un capitaine appelé Crochet (dans cet ordre).
2. Ecrivez dans la classe Militaire une méthode qui affiche un militaire et tous ses supérieurs.
3. Ecrivez une méthode qui recherche dans une structure de type Militaire une personne portant un grade donné. Le grade en question doit être un paramètre de la méthode.