Réflexion & annotations

Exercice 1 - Affichage d'un graphe cyclique

Soit la classe suivante, représentant des graphes orientés avec un noeud initial:

```
public class Graph {
  public Graph(String startName) {
    this.start = new Node(startName);
  }
  public Node getStart() {
    return start;
  }
  public class Node {
    public Node(String name) {
      this.name=name;
    }
    public int hashCode() {
      return name.hashCode();
    }
    public boolean equals(Object o) {
      if (!(o instanceof Node))
        return false;
      return name.equals(((Node)o).name);
    }
    public void add(Node dest) {
      transitions.add(dest);
    }
    public Iterator<Node> transitions() {
      return transitions.iterator();
    }
    private final String name;
    private final ArrayList<Node> transitions =
      new ArrayList<Node>();
  }
  private final Node start;
}
```

On souhaite créer un graphe sachant que le noeud n1 possède deux fils n2 et n3, que le noeud n2 possède un fils n4, que le noeud n4 possède des arcs vers les noeuds n1, n3 et n5 et que n5 possède un lien sur lui-même.

1 Dessiner le graphe correspondant (sur une feuille de papier).

2 Ecrire le code de création du graphe.

3 On souhaite écrire une méthode `toString()` dont l'affichage indique le noeud initial, puis une ligne par noeud du graphe, chaque ligne contenant un identificateur pour le noeud suivi de deux points et des noeuds vers lesquels il possède un arc.

```
Noeud initial : n1
n1: n2 n3
n2: n4
n3:
n4: n3 n1 n5
n5: n5
```

Quel est le parcours de graphe utilisé ?

Quel est la structure de données que l'on doit utiliser pour réaliser ce parcours. A quel interface cela correspond en Java ? Quel implantation choisir ?

Réaliser l'implantation de `toString()`.

4 Ecrire une méthode qui teste si deux noeuds ont le même nom.
Que se passe-t'il s'il y a un cycle ?
Modifier le code sans changer le classe `Node`.

Exercice 2 - Annotation

Déclarer une annotation `@Marked` avec un attribut de type int appelé `level` dont la valeur par défaut est 2

1 Écrire une fonction `printMarked(int level)` qui affiche l'ensemble des champs et méthodes (tous même les pas visibles et ceux hérités) annotés par une annotation `@Marked` dont l'attribut `level` est au moins `level`.

2 Déclarer une annotation `Print` tel que pour un objet
   Si un champs est marqué Print ou Print(true) alors le champs est affiché si c'est un type primitif ou String
   Si un champs est marqué Print ou Print(true) et si son type est autre alors l'ensemble des champs de l'objet est récusivement affiché.
   Si un champs est marqué Print(false) ou n'est pas marqué par Print alors le champs ni affiché ni récursivement parcouru.
   L'annotation ne doit être positionnable que sur les champs.

3 Écrire une fonction `deepPrint(Object o)` qui affiche récursivement les champs de l'objet en utilisant l'annotation Print.
Attention aux cycles.

Exercice 3 - Bidouillage des itérateurs d'ArrayList

Les itérateurs obtenus à partir des collections du paquetage `java.util` sont dits fail-fast, ce qui signifie qu'une modification de la structure itérée entrelacée avec l'utilisation de l'itérateur provoque par cette dernière la levée d'une `ConcurrentModificationException`. Pour contrôler ces modifications, les itérateurs utilisent le champ `modCount` protégé de la classe abstraite `AbstractList`.
Pour vous en convaincre, vous pouvez regarder le code source avec eclipse.
En utilisant les mécanismes de réflexion offerts par le paquetage `java.lang.reflect`, faites en sorte que, dans le cas particulier l'exemple suivant, l'exception `ConcurrentModificationException` ne soit pas levée par l'itérateur d'un `ArrayList`, malgré la modification faite sur cette structure. Pour celà, vous devrez modifier par réflexion la valeur du champ `modCount` de la liste.

```
ArrayList<String> list=new ArrayList<String>();
Collections.addAll(list,args);
for(String s:list) {
  if (s.length()%2==0)
    list.add(s+" stop");
}
```

Exercice 4 - Chargeur de classes prolixe

1 On cherche à écrire un classloader affichant les classes chargées en utilisant les classes de l'exercice 1.
Pour cela, redéfinissez la méthode `loadClass` afin que la méthode affiche les classes

chargées.

```
ClassLoader loader=new ClassLoader() {
  ...
};
loader.loadClass("fr.umlv.java.td7.Graph");
```

2 Appeler par réflexion la méthode `main` de la classe `Graph`.

3 Expliquer pourquoi le classloader n'affiche pas le chargement de la classe `Node` ?

4 Rappeler la différence entre la méthode `findClass` et `loadClass` de la classe `ClassLoader`.
Corrigé votre code en conséquence.

5 Voici le code pour charger une classe sur le disque, en fait il marche pour les classes chargées à partir de n'importe quel URL. On récupère l'URL en utilisant la méthode `getResource(String)`.

```
URLConnection con=url.openConnection();
    byte[] datas=new byte[con.getContentLength()];

    DataInputStream stream=
      new DataInputStream(con.getInputStream());

stream.readFully(datas);
```

Modifier votre classloader pour qu'il charge lui-même les classes de l'exercice 1 en changeant `loadClass`.
Attention, vérifier que vous ne chargez pas plusieurs fois la même classe. De plus, les classes du boot classpath (par exemple `java.lang.String`) ne peuvent pas être chargé par un autre `ClassLoader` que le `ClassLoader` primordiale.

6 Modifier le code pour qu'il utilise un logger (paquetage `java.util.logging.Logger`).