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(Node start) {
    this.start = 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;
}
```

1 Ecrire une méthode `toString()`. 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
```

2 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 paer défaut est 2
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`.

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 Rappeler la différence entre la méthode `findClass` et `loadClass` de la classe `ClassLoader`

2 Compléter le code suivant pour que le classloader affiche lors du chargement la classe de l'exercice 1

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

3 Appeler par reflexion la méthode `main` de la classe `Graph`

4 Expliquer pourquoi le classloader ne charge pas la classe `Node` ?

5 Voici le code pour charger une classe sur le disque. 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 aussi la classe `Node`.
Attention, vérifier que vous ne chargez pas plusieurs fois la même classe. De plus, les classes des paquetages `java.**` ne peuvent pas être chargé par un autre `ClassLoader` que le `ClassLoader` système.

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