programmation
TP N° 8

Visualiseur UML de classes Java.

Le but du travail pratique est de visualiser à la façon UML, un ensemble de classes Java. Pour réaliser ce travail on utilisera le composant graphique *reflexion/PanneauUML* fourni dans la palette de composants enseignement. Ce composant possède une méthode *reAffiche()* qui permet de réafficher le schéma UML, lorsque la liste d'arbres a été modifiée. La première partie du TP consiste à fabriquer cette liste d'arbres.

Première partie.

A partir d'une liste de fichier .class, on charge ces classes dans un *ArrayList*. Chaque classe va être représentée par une instance de la classe *UneClasse* créée par un *new UneClasse(c, new VueClasse(c)).*

Une fois toutes les classes chargées, on met à jour les attributs *laSuperClasse*, *lesSousClasses*, *lesInterfaces*, *laClasseDeclarante*.

On affichera le résultat obtenu dans un *JTree* qui pour chaque classe chargée visualise sa super classes et ses sous classes.

Puis on supprime du tableau toutes les classes qui ne sont pas racines d'une arborescence de dérivation (éventuellement vide).

```
public class UneClasse extends Object {
    public Class laClasse, laSuperClasse;
    public Vector lesSousClasses ;
    public Vector lesInterfaces;
    public VueClasse laVue;
    public UneClasse laClasseDeclarante;

    public UneClasse(Class c, VueClasse lv) {
        laClasse = c;
        laSuperClasse = null;
        lesSousClasses = new Vector();
        laVue = lv;
        lesInterfaces = new Vector();
        laClasseDeclarante = null;
    }
    public boolean equals( Object o){
        if(o instanceof UneClasse){
            UneClasse uc = (UneClasse)o;
            if (laClasse != null)
             return laClasse.getName().equals
                   (uc.laClasse.getName());
            else return false;
        }else   return false;
    }
}
```

On peut alors afficher le résultat par un appel de *reAffiche()*

Deuxième partie.

La deuxième partie consiste à créer une nouvelle classe dérivée de *VueClasse* pour obtenir un affichage des classes qui ne soit pas réduit à leur nom, mais contienne la liste des attributs, la liste des constructeurs, et la liste des mèthodes. La classe *VueClasse* est définie de la façon suivante :

```
public class VueClasse  {
      protected Class laClasse;
      protected Field[] lesAttributs;
      protected Method[] lesMethodes;
      protected Constructor[] lesConstructeurs;
      protected String[] lesAttributsChaine, lesMethodesChaine,
                         lesConstructeursChaine;
      protected int[] lesAttributsMod, lesMethodesMod,
                      lesConstructeursMod;

      protected String nomClasse;
      protected Font fNormale = new Font("Serif", Font.PLAIN, 14);
      protected Font fItalique = new Font("Serif", Font.ITALIC, 14);
      protected Color couleurInterface = new Color(255, 255, 240);
      protected VueClasse laVueDeclarante;
      protected Rectangle bornes;
      protected boolean afficheAPriv, afficheAPub,
                        afficheAPac, afficheAPro;
      protected boolean afficheCPriv, afficheCPub,
                        afficheCPac, afficheCPro;
      protected boolean afficheMPriv, afficheMPub,
                        afficheMPac, afficheMPro;

      public VueClasse(Class c)
      public void calcul(Graphics g)
      public void affiche(Graphics g)
      protected String nomSansPaquetage( String s)
}
```

L'attribut *bornes* donne la position et la taille de l'instance de *VueClasse*.

Les attributs *afficheXY* ou *X* peut être *A* pour attribut, *C* pour constructeur ou *M* pour méthode, et *Y* peut être *Priv* pour privé, *Pub* pour public, *Pac* pour Paquetage, et Pro pour protégé, indiquent qu'on affiche tous les attributs (constructeurs ou méthodes) privés (public, paquetage ou protégés)

La méthode calcul calcule l'attribut *bornes* d'une *VueClasse* sans afficher la vue.

La méthode *nomSansPaquetage (String nom)* calcule un nom sans le nom du paquetage dans lequel il est défini.