Architecture Logicielle – TP 09 2016–2017

Le patron *Proxy*

Exercice 1 – *Proxy de chargement à la demande*

On s'intéresse à l'implémentation d'un navigateur web dans lequel les images incorporées dans les pages seront prises en charge par la classe suivante :

```
1  public class WebImage implements IWebImage {
2    // url de l'image :
3    private final String _URL;
4
5    // Données factices :
6    private int _data;
7
8    public WebImage(String URL) {
9      _URL=URL;
10
11     // Téléchargement de l'image (simulation) :
12     try {
13       Thread.sleep(1000);
14     } catch (Exception e) {
15       System.out.println("Download error !");
16     }
17     _data=12345;
18   }
19
20   public void display(int length, int height) {
21     // Simulation d'affichage :
22     System.out.println("Je suis "+_URL+" affichée en "+length+"x"+height);
23   }
24 }
```

Bien sûr cette classe ne fait que simuler le téléchargement des images par un delai d'attente.

Question 1.1 : Écrivez une `browser` comportant une méthode `loadPage()` qui simule le chargement des ressources associées à une page Web en instanciant un tableau de 30 images dont les URLs respectives seront "http ://the.web.site/image-*i*.jpg" (où *i* est le numéro de l'image).

Question 1.2 : Écrivez une méthode `showPage()` qui simule le rendu de la partie visible de la page en procédant à l'affichage (disons en *320x200*) des 5 premières images seulement.

Question 1.3 : Écrivez un programme de test appelant ces deux méthodes et exécutez le programme. Combien de temps met le programme à s'exécuter ? Au bout de combien de temps les images apparaissent-elles ?

On constate que le temps de téléchargement nécessaire pour afficher seulement une partie de la page est très important. On souhaite pouvoir afficher efficacement des pages de grandes tailles. Pour cela, on se propose de ne télécharger les images qu'en fonction des parties affichées de la page (à la demande). La classe *WebImage* étant un composant à part entière, on ne souhaite pas la modifier. Un *proxy virtuel* a pour but prendre la place d'un objet (on dit que c'est un *placeholder*[1]), pour en différer l'instanciation coûteuse par exemple comme c'est le cas ici.

1. Ce terme n'a pas de traduction exacte en français : on dit souvent emplacement réservé tandis que la signification est plutôt "qui garde la place de".

Question 1.4 : Grâce à quelle précaution de la part de l'auteur de la classe `WebImage` est-il possible d'appliquer le patron proxy ?
Question 1.5 : Dessinez le schéma UML de ce patron proxy.
Question 1.6 : Écrivez une classe proxy pour la classe *WebImage* qui diffère le téléchargement des images sur le Web.
Question 1.7 : Modifiez et ré-exécutez le programme de test. Quel comportement obtient-on ?

Exercice 2 – Codage d'un proxy copy-on-write

Soit la classe suivante :

```
1  public class Matrix {
2    private int _sizem, _sizen;
3    private int[][] _mat;
4
5    // Constructeur initialisant une matrice n*m avec la valeur val
6    public Matrix(int m, int n, int val) {
7      _sizem=m;
8      _sizen=n;
9      _mat=new int[m][n];
10     for (int i=0; i<_sizem; ++i)
11     for (int j=0; j<_sizen; ++j) _mat[i][j]=val;
12   }
13
14   // Fonction d'acces :
15   public void set(int m, int n, int val) { _mat[m][n]=val; }
16   public int get(int m, int n) { return _mat[m][n]; }
17
18   // Constructeur de recopie :
19   public Matrix(Matrix m) {
20     _sizem=m._sizem;
21     _sizen=m._sizen;
22     _mat=new int[_sizem][_sizen];
23     for (int i=0; i<_sizem; ++i)
24     for (int j=0; j<_sizen; ++j) _mat[i][j]=m._mat[i][j];
25   }
26 }
```

Question 2.1 : Que manque-t-il à la classe *Matrix* pour que le patron proxy lui soit applicable (proprement) ? Corrigez et faites un diagramme avec identification des acteurs.
Question 2.2 : Écrivez une classe proxy pour la classe *Matrix* qui évite les copies inutiles en utilisant le patron de proxy *Copy-On-Write*.