Le patron *Adaptateur*

Exercice 1 – *Conception d'un adaptateur*

Un *tokenizer* est un programme qui découpe une chaîne de caractères en morceaux (tokens). La classe `TokenLib` constitue une bibliothèque de fonctions permettant la manipulation de tokens. Parmi ces fonctions, la méthode `displayTokens` permet d'afficher une liste de tokens :

```
1  package tokenizer;
2  import java.util.*;
3  import java.lang.*;
4
5  public class TokenLib {
6    //..
7    public static void displayTokens(Iterator<String> it) {
8      int token_num=0;
9      while (it.hasNext()) {
10       String token=it.next();
11       System.out.println(token_num+" :"+token);
12       token_num++;
13     }
14   }
15   //..
16 }
```

Le programme suivant démontre son utilisation en construisant une telle liste token par token :

```
1  package tokenizer;
2  import java.util.*;
3  import java.lang.*;
4
5  public class Test {
6    public static void main(String[] args) {
7      LinkedList<String> tokenList=new LinkedList<String>();
8      tokenList.add("Ceci");
9      tokenList.add("est");
10     tokenList.add("une");
11     tokenList.add("liste");
12     tokenList.add("de");
13     tokenList.add("tokens");
14     Iterator<String> it=tokenList.iterator();
15     TokenLib.displayTokens(it);
16   }
17 }
```

On souhaite remplacer le code ci-dessus par une tokenisation automatique[1] qui découpera une unique chaîne en fonction d'un ensemble de délimiteurs constitués des espaces, tabulations et retours de ligne. Pour cela, on propose d'ajouter à `TokenLib` une méthode `tokenize()` avec la signature suivante :

```
1  iterator<String> tokenize(String str);
```

Le même programme de test s'écrirait alors :

1. La tokenisation est un processus de démarcation et éventuellement de classification des sections d'une chaîne de caractères. Ici il s'agit de découper la chaîne en utilisant les caractères d'espacement (voir la javadoc pour plus d'options).

```java
package tokenizer;
import java.util.*;
import java.lang.*;

public class Test {
  public static void main(String[] args) {
    Iterator<String> it=TokenLib.tokenize("Ceci est une liste de tokens");
    TokenLib.displayTokens(it);
  }
}
```

On remarque que la classe `java.util.StringTokenizer` réalise exactement ce découpage d'une chaîne de caractères en tokens :

```java
StringTokenizer st = new StringTokenizer("Ceci est un texte a decouper");
```

Note : on souhaite bien évidemment continuer à utiliser la bibliothèque `TokenLib` pour l'affichage et la manipulation de la liste de tokens produite.

Question 1.1 : Recherchez et lisez la documentation javadoc de l'interface *Iterator* sur internet. Attention ! Vérifiez que vous consultez bien la documentation version actuelle (ou ultérieure à *Java 2 Platform SE 5.0*) !

Question 1.2 : Maintenant identifiez quelle interface implémentée par *StringTokenizer* est la plus proche de l'interface *Iterator*. Recherchez et lisez la documentation javadoc de cette interface.

Question 1.3 : Rappelez les différents types d'adaptateurs vus en cours. Lesquels sont compatibles avec le programme à réaliser ?

Question 1.4 : Lequel allez-vous choisir d'implémenter ? Argumentez.

Question 1.5 : Dessinez le diagramme *UML* correspondant à votre solution en identifiant les participants (nom des acteurs du patron) à coté de chaque entité.

Question 1.6 : Programmez et testez l'adaptateur correspondant.

Exercice 2 – *Visualiseur de liste de tokens*

Nous allons reprendre les travaux réalisés à l'exercice précédent et procéder à l'affichage graphique d'une liste de tokens sur deux colonnes de façon similaire à `TokenLib.displayTokens()` (la première sera le numéro séquentiel du token, la second le token). La classe suivante permet d'ouvrir une fenêtre et d'y afficher une table. Le constructeur attend en paramètre un objet dont la classe implémente l'interface `javax.swing.table.TableModel` :

```java
import java.util.*;
import java.lang.*;
import javax.swing.*;
import java.awt.*;
import javax.swing.table.*;

public class TableViewer {
    // Display a Swing component.
    public static void display(Component c, String title) {
        JFrame frame = new JFrame(title);
        frame.getContentPane().add(c);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }

  public TableViewer(TableModel tm) {
        JTable table = new JTable(
        new AbstractTableModel() {
        public int getColumnCount() { return 10; }
        public int getRowCount() { return 10;}
        public Object getValueAt(int row, int col) { return new Integer(row*col); }
     });
        JScrollPane pane = new JScrollPane(table);
        pane.setPreferredSize(new java.awt.Dimension(300, 200));
```

```
26            display(pane, "Token Viewer");
27
28    }
29  }
```

Question 2.1 : Que devez-vous fournir pour faire en sorte que ce composant graphique soit en mesure d'exploiter une liste de tokens ?
Question 2.2 : Lisez la documentation javadoc de l'interface `TableModel` et rappelez le rôle de la classe `AbstractTableModel`. De quel type d'adaptateur s'agira-t-il si vous décidez de l'utiliser ?
Question 2.3 : Donnez le diagramme UML de votre solution annoté avec le nom des acteurs.
Question 2.4 : Programmez l'adaptateur correspondant.

Exercice 3 – *Évolution*
Une première version d'un jeu video a été conçu pour être jouable à la « manette de la joie »(joystick) dont voici l'interface :

```
1  public interface Manette { /* interface pour tous les types de manettes */
2    public boolean isLeft();
3    public boolean isRight();
4    public boolean isPush();
5  }
```

Nous désirons mettre à jour le jeu pour permettre également le jeu au clavier, avec le minimum d'impact sur le code existant. Le clavier sera représenté par la classe suivante :

```
1  public class Clavier {
2    public enum Key { SPACEBAR, ARROW_LEFT, ARROW_RIGHT, /* etc */ };
3    public Clavier() { /* constructeur */ }
4    public Key keyPressed() { /* retourne le type de la touche */ }
5    /* etc */
6  }
```

La classe suivante représente la boucle principale du jeu :

```
1  public class Jeu {
2    private Manette manette;
3
4    public Jeu(Manette manette) { this.manette = manette; }
5
6    public void mainLoop()       { /* boucle principale du jeu */
7      if(manette.isLeft()   { /* deplacement a gauche */ }
8      else if(manette.isRight() { /* deplacement a droite */ }
9      else if(manette.isPush()) { /* appui sur le bouton */}
10   }
11 }
```

Le programme de test pourrait être :

```
1  public class JeuMain {
2    public static void main(String[] args) {
3      Manette manette = new XXXSuperControleur(); /* un certain type de manette */
4      Jeu jeu = new Jeu(manette);
5      jeu.mainLoop();
6    }
7  }
```

Question 3.1 : Donnez le diagramme de classes (vision fournisseur) du jeu vidéo.
Question 3.2 : Proposez une solution pour pouvoir jouer au clavier, sans modifier de classe (à l'exception du main), en utilisant un patron *Adaptateur*.
Question 3.3 : Donnez le code de la classe d'adaptation proposée : *ManetteClavier*[2]. Que doit-on modifier dans le code existant ?

2. C'est une bonne idée, dans le cas du patron adaptateur de nommer la classe par concaténation des noms des interfaces