Le patron *Adaptateur*

Exercice 1 – *Codage d'un adaptateur*

Soit le programme suivant :

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

public class Test {
        public static void afficheTokens(Iterator<String> it) {
                int token_num=0;
                while (it.hasNext()) {
                        String token=it.next();
                        System.out.println(token_num+":"+token);
                        token_num++;
                }
        }

        public static void main(String[] args) {
                LinkedList<String> tokenList=new LinkedList<String>();
                tokenList.add("Ceci");
                tokenList.add("est");
                tokenList.add("une");
                tokenList.add("liste");
                tokenList.add("de");
                tokenList.add("Tokens");
                afficheTokens(tokenList.iterator());
        }
}
```

Ce programme utilise du code hérité (la méthode *afficheTokens*) que l'on ne souhaite pas modifier. Cependant, on souhaiterait pouvoir l'utiliser pour afficher le contenu d'un tokenizer [1] :

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

mais la méthode *afficheTokens* attend en argument un objet présentant une interface d'itérateur.

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

Question 1.2 : Maintenant identifiez quelle interface implémentée par *StringTokenizer* est la plus proche de l'interface *Iterator*.

Question 1.3 : Recherchez et lisez la documentation javadoc de cette interface.

Question 1.4 : Étant donné la structure de ce petit programme, quel type d'adaptateur proposez-vous afin de pouvoir utiliser le résultat du tokenizer avec la méthode *afficheTokens*) ?

Question 1.5 : Dessinez le diagramme *UML* correspondant à votre solution en identifiant les participants à coté de chaque entité.

Question 1.6 : Écrivez l'adaptateur correspondant.

Exercice 2 – *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 :

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

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).

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 :

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

La classe suivante représente la boucle principale du jeu :

```
public class Jeu {
        private Manette manette;

        public Jeu(Manette manette) { this.manette = manette; }

        public void mainLoop()                  { /* boucle principale du jeu */
                if(manette.isLeft()             { /* deplacement a gauche */ }
                else if(manette.isRight()       { /* deplacement a droite */ }
                else if(manette.isPush())       { /* appui sur le bouton */}
        }
}
```

Le programme de test pourrait être :

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

Question 2.1 : Donnez le diagramme de classes (vision fournisseur) du jeu vidéo.
Question 2.2 : Proposez une solution pour pouvoir jouer au clavier, sans modifier de classe (à l'exception du main), en utilisant un patron *Adaptateur*.
Question 2.3 : 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