

Exercices de Patterns (2)

1. On désire écrire un petit logiciel permettant de lister les mots présents dans un fichier et répondant à une certaine condition. Une approche naïve a produit les programmes suivants :

Le premier permet de lister les mots commençant par 't' :

```
import java.io.*;
import java.util.*;

public class ListerMots1 { // mal foutu
    private String fichier;

    public ListerMots1(String fichier) {
        this.fichier = fichier;
    }

    public void imprimerSiCommenceParT() throws IOException {
        BufferedReader input = new BufferedReader(
            new FileReader(this.fichier));
        String buffer = null;
        while ((buffer = input.readLine()) != null) {
            StringTokenizer mots = new StringTokenizer(buffer,
                " \t.;(){}\"'*=:/\\");
            while (mots.hasMoreTokens()) {
                String mot = mots.nextToken();
                if (mot.charAt(0) == 't')
                    System.out.println(mot);
            }
        }
    }

    public static void main(String[] args) throws IOException {
        if (args.length == 0) {
            System.out.println("Usage : java ListerMots1 fichier");
            System.exit(1);
        }
        new ListerMots1(args[0]).imprimerSiCommenceParT();
    }
}
```

Un deuxième programme permet de lister les mots d'une longueur précisée :

```
import java.io.*;
import java.util.*;

public class ListerMots2 { // mal foutu
    private String fichier;

    public ListerMots2(String fichier) {
        this.fichier = fichier;
    }

    public void imprimerSiDeLongueur(int longueur) throws IOException {
        BufferedReader input = new BufferedReader(
            new FileReader(this.fichier));
        String buffer = null;
        while ((buffer = input.readLine()) != null) {
            StringTokenizer mots = new StringTokenizer(buffer,
                " \t.;(){}\"'*=:/\\");
            while (mots.hasMoreTokens()) {
                String mot = mots.nextToken();
                if (mot.length() == longueur)
                    System.out.println(mot);
            }
        }
    }
}
```

Exercices de Patterns (2)

```
    }  
    }  
}  
  
public static void main(String[] args) throws IOException {  
    if (args.length == 0) {  
        System.out.println("Usage : java ListerMots3 fichier");  
        System.exit(1);  
    }  
    new ListerMots2(args[0]).imprimerSiDeLongueur(5);  
}
```

Un troisième permet de lister les palindromes :

```
import java.io.*;  
import java.util.StringTokenizer;  
  
public class ListerMots3 { // mal foutu  
  
    private String fichier;  
  
    public ListerMots3(String fichier) {  
        this.fichier = fichier;  
    }  
  
    public void imprimerPalindromes() throws IOException {  
        BufferedReader input = new BufferedReader(  
            new FileReader(this.fichier));  
  
        String buffer = null;  
        while ((buffer = input.readLine()) != null) {  
            StringTokenizer mots = new StringTokenizer(buffer,  
                " \t.;(){}\"'*=:/\\\"");  
            while (mots.hasMoreTokens()) {  
                String mot = mots.nextToken();  
                if (estPalindrome(mot))  
                    System.out.println(mot);  
            }  
        }  
    }  
  
    public boolean estPalindrome(String mot) {  
        if (mot == null)  
            return false;  
        StringBuffer stringbuffer = new StringBuffer(mot);  
        return mot.equals(stringbuffer.reverse().toString());  
    }  
  
    public static void main(String args[]) throws IOException {  
        if (args.length == 0) {  
            System.out.println("Usage : java ListerMots4 fichier");  
            System.exit(1);  
        }  
        new ListerMots3(args[0]).imprimerPalindromes();  
    }  
}
```

On vous demande de modifier ces programmes afin de n'écrire qu'une seule fois le programme de base et de pouvoir choisir la condition **que doivent vérifier les mots listés**. On généralisera aussi le premier programme afin de lister les mots commençant par n'importe quelle chaîne de caractères. Quel pattern utiliserez vous pour cela ?

Exercices de Patterns (2)

2. Jusqu'ici notre application n'a implémenté que des sélections primitives. On désire à présent combiner plusieurs sélections.
 - 2.a. Ecrivez une classe `And` qui permet de combiner deux sélections selon la règle
sélection : sélection AND sélection
 - 2.b. Ecrivez de même une classe `Or` et une classe `Not` en respectant les règles :
sélection : sélection OR sélection
sélection : NOT sélection.
 - 2.c. Quel pattern est utilisé dans cette construction ?
 - 2.d. Donnez la correspondance entre les noms des classes utilisées dans la théorie et celle utilisées dans le code que vous écrirez.
3. Ecrivez à présent une classe `Et` qui combine un nombre quelconque de sélections pour ne sélectionner que les mots qui satisfassent à toutes. Ecrivez semblablement une classe `Ou`.
4. On voudrait à présent que, si l'utilisateur le désire, il puisse connaître le nombre de mots sélectionnés. Il est clair qu'on ne va pas écrire une sous-classe de chaque classe de sélection pour arriver au résultat. Un pattern permet de le faire en n'écrivant qu'une seule classe : la classe `Compteur`.
 - 4.a. Ecrivez cette classe.
 - 4.b. Quel pattern avez-vous utilisé pour résoudre ce problème ?
 - 4.c. Donnez la correspondance entre les noms des classes utilisées dans la théorie et celle utilisées dans votre code.