Génie logiciel

—

TD4

Pietri Yoann (A11)
Levy–Falk Hugo (A11)

Mars 2018

Table des matières

1 Lecture d'une chaîne de caractères dans un fichier

Le mot clé `static` permet d'appeler la méthode `lireTexte` sans avoir besoin d'instancier d'objet de la classe `Utilitaire`.

Code source de la classe Comparateur :

```
package question1;

public class Comparateur {

	public static void main(String[] args) {
		String nomFichier = "texte2.txt";
		String txt = Utilitaire.lireTexte(nomFichier);
		System.out.println(txt);
	}
}
```

2 Implémentation de l'architecture logicielle

2.1 Code source de la classe `Comparateur`

```
package question2;

import java.util.ArrayList;

public class Comparateur {

	public static void main(String[] arg) {
		int TAILLE_EXTRAIT = 50;
		String texteClair = Utilitaire.lireTexte("texte2.txt");
		System.out.println("début du texte original : "
				+ texteClair.substring(0, Math.min(50, texteClair.length())));

		ArrayList<AlgoCodage> algoAComparer = new ArrayList<AlgoCodage>();
		algoAComparer.add(new Cesar(3));
		algoAComparer.add(new Morse());
		algoAComparer.add(new Cesar(10));

		String encodee = "";
		for (AlgoCodage algo : algoAComparer) {
			System.out.println("algorithme : " + algo.getNom());
			encodee = algo.encode(texteClair);
			System.out.println("\tencodage : "
				+ "\t résultat : "
				+ encodee.substring(0, Math.min(50, encodee.length())));

			String decodee = algo.decode(encodee);
			System.out.println("\tdécodage : "
				+ "\t résultat : "
				+ decodee.substring(0, Math.min(50, decodee.length())));
		}
	}
}
```

2.2 Code source de la classe `AlgoCodage`

```
package question2;

public abstract class AlgoCodage {

	protected String nom;
```

```
        public abstract String encode(String s);

        public abstract String decode(String s);

        public String getNom() {
                return nom;
        }


}
```

2.3 Code source de la classe `Cesar`

```
package question2;

public class Cesar extends AlgoCodage{

        private int decalage;
        private String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        public Cesar(int decalage){
                this.nom = "César";
                this.decalage = decalage;
        }

        @Override
        public String encode(String s) {
                String encode = "";
                for(int i=0;i<s.length();i++){
                        encode += alphabet.charAt((decalage +
                                                alphabet.indexOf(s.charAt(i)))%alphabet.length());
                }
                return encode;
        }

        @Override
        public String decode(String s) {
                return "Fonctionnalité décodage de César pas encore implémenté";
        }



}
```

3 Utilisation d'un patron pour ajouter un algorithme existant

Les classe `EncodeurAClef` ne possède pas les méthodes `encode` et `décode`, mais possède des méthodes `chiffre` et `déchiffre`. De plus il n'hérite pas de `AlgoCodage`. La solution est d'utiliser le patron adaptateur.

La figure 1 montre le diagramme UML de l'application.

Le code à ajouter dans la classe comparateur est alors :

```
algoAComparer.add(new AdapteEncodeurAClef("mon secret"));
algoAComparer.add(new AdapteEncodeurAClef(""));
```

4 Tests unitaires pour codage d'Huffman

Code source de la classe `ArbreTest` :

```
package question4;
```

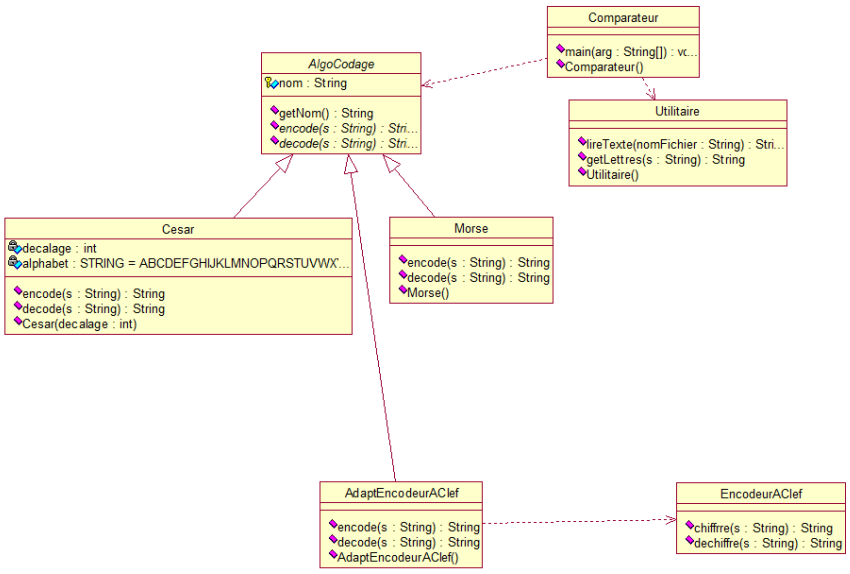

Figure 1 – Diagramme UML

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class ArbreTest {

        Arbre arbre1;
        Arbre arbre2;

        @Before
        public void setup() throws Exception{
                arbre1 = Arbre.buildTree("a");

                arbre2 = Arbre.buildTree("baob");
        }

        @Test
        public void testBuildTree() {
                assertEquals("La lettre n'est pas dans une feuille simple",
                                ((Feuille)arbre1).lettre, 'a');
                assertEquals("La fréquence est mauvaise dans une feuille simple",
                                ((Feuille)arbre1).frequence, 1 );

                assertEquals(arbre2.frequence, 4 );
                Feuille feuilleB = (Feuille)((Noeud)arbre2).filsGauche;
                assertEquals(feuilleB.frequence, 2);
                assertEquals(feuilleB.lettre, 'b');
                Noeud noeudDroit = (Noeud)((Noeud)arbre2).filsDroit;
                assertEquals(noeudDroit.frequence, 2);
                Feuille feuilleA = (Feuille)noeudDroit.filsGauche;
                assertEquals(feuilleA.lettre, 'a');
                assertEquals(feuilleA.frequence, 1);
                Feuille feuilleO = (Feuille)noeudDroit.filsDroit;
                assertEquals(feuilleO.lettre, 'o');
                assertEquals(feuilleO.frequence, 1);
        }
```

```
}
```

5 Documentation

Code source de la classe Feuille :

```
package question5;

/**
 * Cette classe représente une feuille d'un arbre de Huffman.
 * @author galtier
 *
 */
public class Feuille extends Arbre {
	final char lettre;

	/**
	 *
	 * @param frequence nombre d'occurrences de cette lettre dans le générateur
	 * @param lettre caractère dont on compte le nombre d'occurences
	 */
	public Feuille(int frequence, char lettre) {
		super(frequence);
		this.lettre = lettre;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Feuille other = (Feuille) obj;
		if ((lettre != other.lettre) || (this.frequence != other.frequence))
			return false;
		return true;
	}


}
```

Table des figures