ISEN Yncréa Ouest – M1
Langage Java - TP évalué
Durée 3 heures - Documents autorisés

Bien lire tout l'énoncé avant de commencer !

Le Javanais !!!

Introduction

*Le **javanais** ou **langue de feu**, apparu en France dans la dernière moitié du XIX^e^ siècle, est un procédé de codage argotique utilisant une phonologie parasitaire constituée par l'insertion d'une syllabe supplémentaire entre voyelles et consonnes, dans le but de rendre ce texte moins compréhensible aux non-initiés.*

Cette syllabe comporte un son lié au nom de la variante : « ja » ou « av » dans la variante « javanaise » et une syllabe comportant « f » dans la variante « langue de feu ». Extrait de https://fr.wikipedia.org/wiki/Javanais_(argot)

Les règles que nous appliquerons sont les suivantes :

- av est ajouté après chaque consonne (ou groupe de consonnes comme ch, cl, ph, tr,…) d'un mot, autrement dit avant chaque voyelle.
- Si le mot commence par une voyelle, av est ajouté devant cette voyelle.
- av n'est jamais ajouté après la consonne finale d'un mot.
- « y » est traité comme une consonne s'il est suivi d'une voyelle. Par exemple, le mot « moyen » est codé mavoyaven (et non pas mavoavyaven).
- Les monosyllabes (« a », « à », « en », « un ») ainsi que les mots commençant par une voyelle prennent une syllabe supplémentaire initiale. Ainsi, « **a**bricot » est codé **ava**bravicavot.

Quelques exemples :

Train	Travaavain
Bonjour	Bavonjavoavur
Supermarché	Savupavermavarchavé
Etudiant	Avetavudaviavant
Brian is in the kitchen	Braviavan avis avin thave kavitchaven

Travail à réaliser

Vous devez développer une interface utilisateur pour mettre en œuvre un traducteur Français – Javanais.

Votre interface doit comporter, **au moins**, les composants suivants (voir ci-dessous figure 1) :

- une zone « saisie » pour saisir le texte à traduire,
- une zone « affichage » pour afficher la traduction,
- une zone « bouton » pour lancer la traduction, fermer l'application, effacer les zones de saisie et d'affichage.

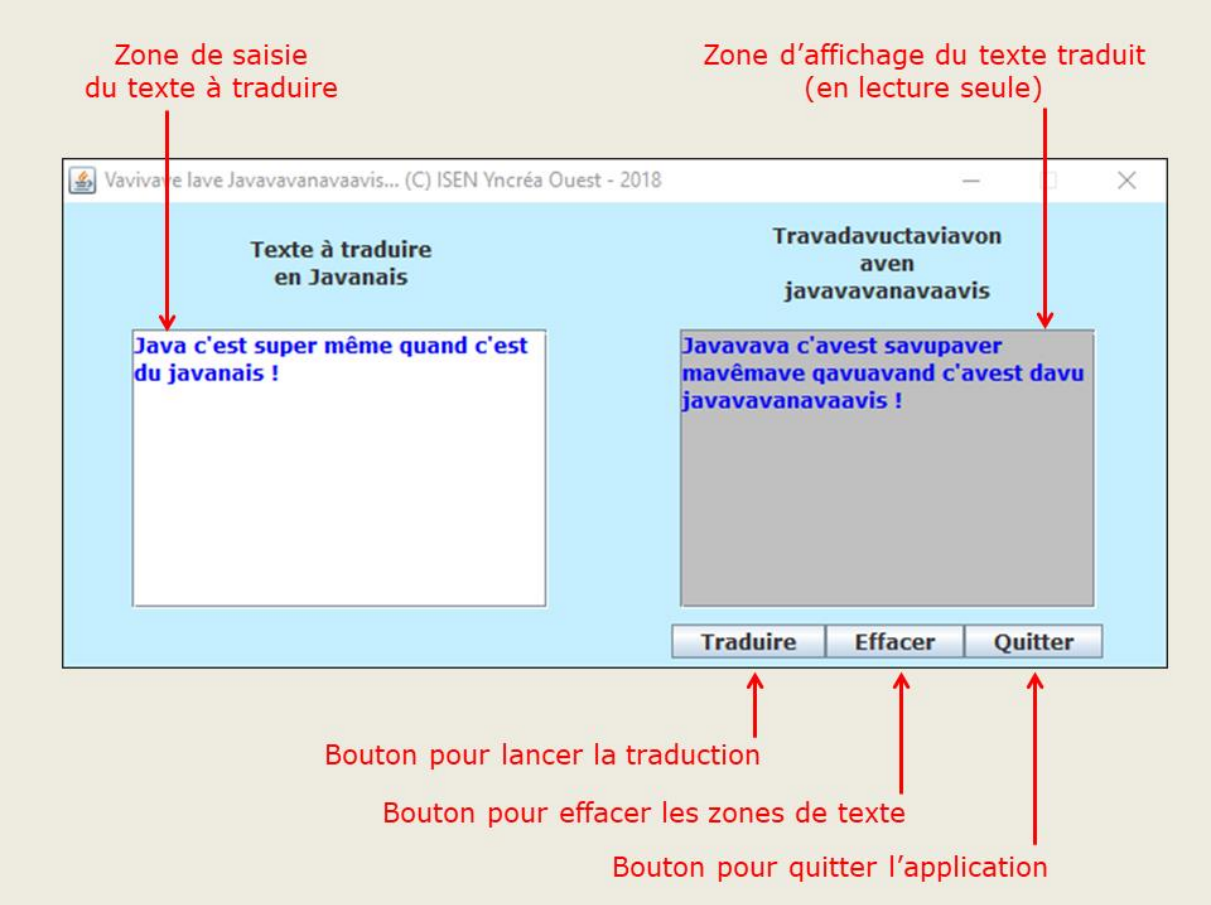

Figure 1. Version minimale de l'interface

Attention, version minimale n'est pas synonyme d'achèvement.
Une partie de l'évaluation portera sur l'ajout de fonctionnalités non précisées dans ce cahier des charges. A vous d'imaginer et de mettre en œuvre…

Dans votre rapport vous expliquerez votre raisonnement pour la mise en œuvre de votre application. Vous détaillerez également son architecture. N'oubliez pas de décrire vos tests !

En fin de TP, vous créerez un fichier d'archive regroupant vos codes sources ainsi que votre rapport. Vous déposerez ce fichier sur l'ENT dans la rubrique ad hoc.

Vos productions seront compilées puis testées sous l'environnement jGrasp par le correcteur.

Une attention particulière sera notamment portée sur :

- le respect du cahier des charges (i.e. le travail minimal à réaliser),
- l'ergonomie de votre application,
- l'ajout de fonctionnalités non précisées dans le cahier des charges,
- la pertinence des « concepts objets » mis en œuvre.

Conseils / Remarques

Pour vous mettre sur la voie on vous propose en annexe la classe Lettre qui comporte quelques méthodes intéressantes…

Si vous utilisez des ressources externes (i.e. sur internet) vous devez mentionner l'origine de ces ressources en référence dans votre rapport et en commentaire dans votre code. Toute omission sera sanctionnée.
On trouve sur internet de nombreux composants logiciels. Ne vous dispersez pas. Ne perdez pas de vue votre cahier des charges.

Annexe 1 : La classe Lettre

```
import java.util.*;

public class Lettre {

   private Character valeur;
   private boolean minuscule;

   private static Character[] voyelles = {'a', 'à', 'â', 'e', 'é', 'è', 'ê', 'ë', 'i', 'î', 'ï', 'o',
'ô', 'u', 'ù', 'û', 'ü', 'y', 'ÿ'};


/**
 * Le constructeur de la classe Lettre.
 *
 * @param   char  Le caractère concerné
 */
   Lettre(char x) {
      if (Character.isLowerCase(x)) {
         this.minuscule = true;
         this.valeur = new Character(x);
      }
      else {
         this.minuscule = false;
         this.valeur = new Character(Character.toLowerCase(x));
      }
   }


/**
 * Cette méthode permet de savoir si la lettre est une voyelle.
 *
 * @return  boolean  Vrai si voyelle, faux sinon
 */
   public boolean estVoyelle() {
      List<Character> list = Arrays.asList(voyelles);
      if (list.contains(this.valeur)) {
         return(true);
      }
      else {
         return(false);
      }
   }
```

```
/**
 * Cette méthode permet de savoir si la lettre est un y.
 *
 * @return  boolean  Vrai si y, faux sinon
 */
  public boolean estY() {
    if ( (this.valeur=='y') || (this.valeur=='ÿ') ) {
      return(true);
    }
    else {
      return(false);
    }
  }

 /**
 * Cette méthode retourne une représentation de l'objet
 * sous forme de chaîne de caractères.
 *
 * @return  String   Chaîne de caractères
 */
  public String toString() {
    char x = this.valeur.charValue();
    if (this.minuscule) {
      return new Character(x).toString();
    }
    else {
      return new Character(Character.toUpperCase(x)).toString();
    }
  }

}
```