Outil de traduction de textes à la Google Translate

L3 MIAGE - Université Paris 1 Panthéon Sorbonne

Carmen Brando

L'objectif de cet exercice est de **développer une application de type client lourd qui sert à la traduction de textes**. Nous allons nous inspirer de l'outil Google Translate[1] de Google, une impression écran de l'interface graphique est présentée en figure 1. L'implémentation de cette application utilise le **modèle Modèle-Vue-Contrôleur** et est implémentée en **JavaFX**.

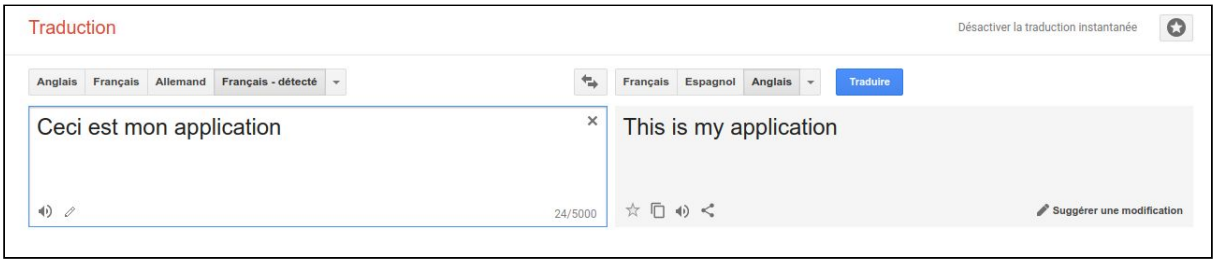

Figure 1 : interface graphique de Google Translate

Afin de développer cette application à partir du modèle MVC, nous devons **créer une classe Java** pour chacun des composants listés ci-dessous.

- la **vue** décrit les **éléments graphiques** qui seront affichés sur l'interface utilisateur,
- le **modèle** explicite les traitements métiers (les spécifications fonctionnelles),
- le **contrôleur** définit le comportement de l'application en gérant les interactions de l'utilisateur sur l'interface graphique,
- il est nécessaire de créer une méthode **main** qui exécute les instructions principales de l'application comme par exemple, l'initialisation de la scène JavaFX.

Ces composantes seront ensuite décrits de manière détaillée, lisez les extraits du code qui vous sont proposés, copiez-les sur votre projet de développement et complétez les instructions manquantes.

1) LE MAIN

Le programme principale est construit au fur et à mesure que nous développons notre application. N'hésitez pas à passer sur les points suivants et revenir ensuite à compléter le main.

[1] http://translate.google.fr

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class App extends Application {

      @Override
    public void start(Stage primaryStage) throws Exception {

        //COMPLÉTEZ LE CODE
      scene.getStylesheets().add(getClass().getResource("styles.css").toExter
nalForm());
      primaryStage.show();
    }

    public static void main(String[] args) {

    }
}
```

Veuillez fournir une description de cette classe quand vous l'aurez finalisée.

2) LA VUE

Pour créer notre interface graphique, nous utiliserons le **langage FXML** qui est fondé sur le langage de balisage XML. Il est nécessaire d'identifier les **éléments graphiques** que l'on souhaite faire apparaître sur l'interface graphique et qui permettront de faire interagir les utilisateurs avec notre application. Ces éléments graphiques peuvent être **les boutons, les champs de texte, les listes déroulantes**, entre autres (vous trouvez une référence détaillée ici[2]). Veuillez observer la figure 1 et répondre aux questions suivantes : Quels type d'éléments graphique vous observez ? Combien d'éléments existent pour chaque type ?

[2] https://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm

Figure 2 : contrôles d'interfaces sous JavaFX

Veuillez noter que pour cette interface, nous allons nous intéresser à un type de boutons un peu particulier appelés ***Toggle buttons***. Ils sont souvent rassemblés dans un *Toggle group* et sont pratiques quand l'on souhaite sélectionner un bouton à la fois ; autrement dit, seul un bouton dans le groupe peut être sélectionné par l'utilisateur à un moment donné.

Les éléments graphiques peuvent aussi être invisibles, c'est le cas des **conteneurs**. Ils permettent uniquement de rassembler les éléments visibles selon un modèle d'agencement (*Layout*) prédéfini et ils peuvent également être composés d'autres conteneur. A partir du texte présenté ci-dessous (Aide) et de la figure 1 : observez-vous des groupes d'éléments ayant une disposition particulière (droite, gauche, horizontal, vertical, ..) ?

Aide.

En JavaFX, il existe huit types de layouts :

- Le **BorderPane** qui vous permet de diviser une zone graphique en cinq parties : top, down, right, left et center.
- La **Hbox** qui vous permet d'aligner horizontalement vos éléments graphiques.
- La **VBox** qui vous permet d'aligner verticalement vos éléments graphiques.
- Le **StackPane** qui vous permet de ranger vos éléments de façon à ce que chaque nouvel élément inséré apparaisse au-dessus de tous les autres.
- Le **GridPane** permet de créer une grille d'éléments organisés en lignes et en colonnes
- Le **FlowPane** permet de ranger des éléments de façon à ce qu'ils se positionnent automatiquement en fonction de leur taille et de celle du layout.
- Le **TilePane** est similaire au FlowPane, chacune de ses cellules fait la même taille.
- L'**AnchorPane** permet de fixer un élément graphique par rapport à un des bords de la fenêtre : top, bottom, right et left.

Le squelette de l'interface graphique en FXML est présenté ci-dessous :

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.VBox?>

<VBox spacing="10" prefWidth="900" prefHeight="300" fx:controller="??"
xmlns:fx="http://javafx.com/fxml">
<fx:define>
      <ToggleGroup fx:id="toggleGroupIn"/>
      <!-- COMPLÉTEZ LE SQUELETTE DU CODE -->
      </fx:define>

      <!-- COMPLÉTEZ LE SQUELETTE DU CODE -->

      <HBox spacing="2">
         <ToggleButton text="Anglais" toggleGroup="$toggleGroupIn"/>
         <ToggleButton text="Français" toggleGroup="$toggleGroupIn"/>
      <!-- COMPLÉTEZ LE SQUELETTE DU CODE -->

      </HBox>
</VBox>
```

Complétez les instructions manquantes au fur et à mesure et décrivez avec vos propres mots ce qui est spécifié dans cette interface.

En outre, JavaFX permet l'utilisation de **CSS** pour **définir le style des éléments graphiques**, vous pouvez récupérer le code ci-dessous et l'incorporer dans votre application.

```
.root {
      -fx-background-color: white;
}
.button {
      -fx-background-color: blue;
      -fx-text-fill: white;
}
.label {
```

```
    -fx-text-fill: red;
    -fx-font-size: 18;
}
```

3) LE MODELE

Le modèle tout simplement est en charge de lancer les méthodes spécifiques à la traduction de texte. Vous n'avez pas besoin d'implémenter un algorithme de traduction de textes, vous pouvez compléter le code à la mesure de vos envies.

```
public class Model {

    public String translate(String text, String inLan, String outLan) {

        return "bonjour"
        //COMPLÉTEZ LE SQUELETTE DU CODE
    }

}
```

4) LE CONTROLEUR

Le contrôleur doit être capable de recevoir les entrées de l'utilisateur via les contrôles définis sur la vue, à savoir, le texte à traduire, la langue de ce texte et la langue de sortie. Cette application possède une seule fonctionnalité, celle de traduire, il est donc logique d'a

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;

public class Controller {

    @FXML
    private ToggleGroup toggleGroupIn;

    @FXML
    private ToggleGroup toggleGroupOut;
```

```
        private Model model = new Model();

        //COMPLÉTEZ LE CODE

        @FXML
        private void processTranslation(ActionEvent event) {
                String inLan = ((ToggleButton)
toggleGroupIn.getSelectedToggle()).getText();
                String outLan = ((ToggleButton)
toggleGroupOut.getSelectedToggle()).getText();

                //COMPLÉTEZ LE CODE
        }
}
```

La version finale de l'application est présentée en figure 3.

Figure 3 : client lourd de traduction de textes

Développez ensuite les fonctionnalités suivantes :

5) Ajoutez une **liste déroulante** afin que l'utilisateur puisse sélectionner une langue quelconque parmi un vingtaine en tant que langue d'origine et en tant que langue de sortie
6) Ajoutez un bouton pour permettre l'utilisateur de charger un fichier .TXT depuis son ordinateur afin qu'il soit traité en tant que texte d'entrée à traduire