TP4

Java FX 8 : Notre première UI.

Nous allons essayer de modéliser au mieux l'UI du TD2 :

Nous allons y aller pas à pas. Et nous arrêterons juste avant l'implémentation des checklists.

MVC

Le code suivant met en place le pattern de conception MVC :

- le Modèle est pour l'instant une `ArrayList<String>`
- la Vue : `UserInterface.java`
- Le contrôleur : `Controller.java`

Cette construction est pour l'instant assez vide mais elle permet de voir :

- une organisation générale en MVC avec trois classes distinctes.
- pour la vue :
  - une façon d'organiser les éléments horizontalement avec HBox

    - les éléments d'interface TextField (pour écrire une ligne) et Button pour créer un bouton.
- pour le contrôleur :
    - le constructeur qui lie modèle et vue et prépare les réactions aux événements.
    - lier l'action d'un bouton à une méthode du contrôleur (même si pour l'instant le bouton n'est pas celui de la vue...)

Le code

Trois classes : Main, UserInterface et Controller

Main.java

```
package com.mco;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.util.ArrayList;

public class Main extends Application {

    public static void main(String[] args) {
        launch(args);
    }


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

        //Modèle
        ArrayList<String> todoList = new ArrayList<>();

        //Vue
        UserInterface userInterface = new UserInterface();

        //Contrôleur
        Controller controller = new Controller(todoList, userInterface);

        //Affichage de l'UI
        Scene theScene = new Scene(userInterface.getRoot());
        primaryStage.setScene(theScene);
        primaryStage.setTitle("Todo List");
        primaryStage.show();
    }
}
```

UserInterface.java

```
package com.mco;

import javafx.scene.Group;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.text.Text;

public class UserInterface {
    private Group root;
    private Button addButton;

    public UserInterface() {
        root = new Group();
        generateUI();
    }

    private void generateUI() {

        //First block
        HBox hBox = new HBox(10);
        Text text = new Text("Input:");
        TextField textField = new TextField();
        addButton = new Button("add");
        hBox.getChildren().addAll(text, textField, addButton);

        root.getChildren().add(hBox);

    }

    public Group getRoot() {
        return root;
    }

    public Button getAddButton() {
        return addButton;
    }
}
```

Controller.java

```
package com.mco;
```

```
import javafx.event.ActionEvent;
import javafx.scene.control.Button;

import java.util.ArrayList;

public class Controller {
    private ArrayList<String> model;
    private UserInterface vue;

    public Controller(ArrayList<String> model, UserInterface vue) {
        this.model = model;
        this.vue = vue;


        vue.getAddButton().setOnAction(this::handleButtonAdd);
    }

    private void handleButtonAdd(ActionEvent actionEvent) {
        System.out.println("Click !");
    }
}
```

A faire

1. Créez un projet et faites en sorte d'exécuter le code précédent (il faut 3 classes, donc 3 fichiers et copier/coller les contenus)
2. Comprenez comment le code fonctionne. En particulier les liens entre les objets de la vue et du contrôleur.
3. Faites en sorte que soit affiché dans la console le texte duTextField lorsque l'on clique sur le bouton (comment trouver le texte affiché dans un TextField ?)
4. Plutôt que de l'afficher, ajoutez le texte au modèle et affichez le modèle.
5. N'ajoutez au modèle que les chaînes de caractères non vides.

Amélioration de UI

Javafx 8 permet de faire de nombreux agencements de composants graphique, vous pourrez en voir la plupart dans ce tutorial sur les layouts. Nous n'utiliserons qu'une petite partie ce ces possibilités.

Représentez le modèle

1. En utilisant des VBox (Vertical box) faites en sorte d'ajouter chaque item à la vue (un Text par exemple). Il pourra être nécessaire d'agrandir la fenêtre pour voir les ajouts.
2. Insérez votre liste dans un ScrollPane pour que l'on puisse voir des listes de taille très grande.

Ajout du résumé

Ajoutez une troisième partie à votre UI contenant le résumé. Pour l'instant ce résumé ne doit contenir que ne nombre d'items du modèle.

Pour cela il pourra être nécessaire de :

- créer un attribut pour le nombre d'items
- créer un setter pour le nombre d'items
- modifier la valeur du texte dans la vue lorsque ce nombre change
- transformer un nombre en chaîne de caractères en utilisant la méthode `String.valueOf()`

Mise à jour du modèle

Un todoItem est maintenant composé non seulement d'un intitulé, mais également de sa date de création.

A faire

1. Créez une classe TodoItem composée d'un intitulé texte et d'une date (on pourra utiliser la classe Date de java). Le constructeur doit prendre comme paramètre le nom de l'item et placer la date de création à la date courante.
2. Mettez à jour le modèle pour qu'il devienne une `ArrayList<TodoItem>`. Adaptez le contrôleur au nouveau modèle de données. Pour l'instant ne prenez pas en compte la Date.
3. Ajoutez la Date à la représentation graphique. Vous pourrez pour cela ajouter une méthode à la vue qui ajoute un TodoItem à la vue `void addTodoItem(TodoItem item)`. Plutôt que d'ajouter simplement du texte, on pourra ajouter une HBox contenant le texte et la date sous forme de texte (prenez celle par défaut pour l'instant).

Mise à jour de la vue

Plus on va ajouter des choses à la vue, moins la Classe UserInterface sera compréhensible.

A faire

Séparez les parties de la classe UserInterface en plusieurs classes. On conservera UserInterface qui fera le lien entre les parties et le contrôleur.

Créer les classes :

- `InputUI` : qui sera en charge de l'intitulé de l'item à ajouter et du bouton Add
- `TodoUI` : qui gérera la liste des items sous forme graphique
- `SummaryUI` : qui contiendra le résumé.

Vous devrez résoudre les problèmes suivants :

- les sous-classes vont fabriquer leur propre arbre de scène qu'il faudra ajouter à l'arbre de l'interface globale

- les liens vers les objets de l'interface utiles au contrôleur doivent être conservés.