R2.02 – IHM
Correction du TP n°1 & Introduction au TP n°2
Interface simple en JavaFX – Calcul de la moyenne

IUT Nord Franche-Comté

Université Marie-et-Louis-Pasteur

2025–2026

Sommaire

Qu'est-ce que JavaFX ?

- **JavaFX** est une bibliothèque graphique pour créer des interfaces utilisateur (GUI) en Java.
- Elle remplace l'ancienne bibliothèque `Swing`.
- Depuis Java 11, JavaFX est distribuée séparément (SDK à télécharger sur `openjfx.io`).

Architecture d'une application JavaFX

`Stage`	La fenêtre principale (fournie par le système).
`Scene`	Le conteneur racine qui contient tous les éléments graphiques.
`Node`	Tout élément visuel : boutons, labels, champs de texte, layouts...

Structure minimale d'une application JavaFX

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        // 1. Créer les widgets
        // 2. Les placer dans un layout
        // 3. Créer la Scene avec le layout
        // 4. Configurer et afficher le Stage

        primaryStage.setTitle("Mon application");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);  // Démarre l'application JavaFX
    }
}
```

Important

La classe doit hériter de `Application` et redéfinir la méthode `start()`.

Les composants graphiques (Widgets)

Saisie de données :

`TextField` Champ de saisie texte (une ligne). Utilisé ici pour entrer les notes.

`RadioButton` Bouton radio – un seul choix possible dans un groupe. Utilisé pour les coefficients (1, 2, 3).

`ToggleGroup` Regroupe des `RadioButton` pour qu'un seul soit sélectionné à la fois.

Autres composants :

`ComboBox` Liste déroulante (Latin, Grec, Sport).

`CheckBox` Case à cocher (coefficients oui/non).

`Button` Bouton cliquable (« Moyenne »).

`Label` Texte statique non modifiable par l'utilisateur.

Les attributs de la classe `Main`

Règle : seuls les widgets avec lesquels on interagit (lire ou modifier leur état) sont déclarés comme attributs de classe.

Attributs de classe

- `TextField` (notes à lire)
- `RadioButton` (coef. à lire)
- `ToggleGroup` (groupes radio)
- `ComboBox` (option à lire)
- `CheckBox` (état à lire)
- `Button` (action à associer)
- `Label lMoy` (résultat à modifier)

PAS attributs de classe

Les `Label` statiques comme « Anglais », « Mathématiques », « Optionnel » ne sont **pas** des attributs car on ne les modifie jamais.

Ils sont créés directement dans les méthodes de mise en page.

Déclaration des attributs

```
public class Main extends Application {

    // TextFields pour les notes
    private TextField tfAnglais, tfMaths, tfInfo, tfGeo, tfOption;

    // RadioButtons pour les coefficients
    private RadioButton rbAng1, rbAng2, rbAng3;
    private RadioButton rbMaths1, rbMaths2, rbMaths3;
    // ... idem pour Info et Géo

    // ToggleGroups (un par matière)
    private ToggleGroup tgAnglais, tgMaths, tgInfo, tgGeo;

    // Autres composants
    private ComboBox<String> listOption;
    private CheckBox cbCoef;
    private Button btnMoyenne;
    private Label lMoy;
```

Création des widgets – `initWidgets()`

Principe

Cette méthode crée et initialise **tous** les widgets, mais ne les place dans **aucun** layout. La séparation création / placement rend le code plus lisible.

```
private void initWidgets() {
    tfAnglais = new TextField();
    tfMaths   = new TextField();
    // ...

    // RadioButtons + ToggleGroup pour Anglais
    tgAnglais = new ToggleGroup();
    rbAng1 = new RadioButton("1");
    rbAng2 = new RadioButton("2");
    rbAng3 = new RadioButton("3");
    rbAng1.setToggleGroup(tgAnglais);
    rbAng2.setToggleGroup(tgAnglais);
    rbAng3.setToggleGroup(tgAnglais);
    rbAng1.setSelected(true); // Sélection par défaut
    // ... idem pour chaque matière
}
```

initWidgets() – suite

```
// ComboBox pour l'option
listOption = new ComboBox<>();
listOption.getItems().addAll("Latin", "Grec", "Sport");
listOption.setValue("Latin"); // Valeur par défaut

// CheckBox
cbCoef = new CheckBox(
    "Prendre en considération les coefficients");

// Bouton et label résultat
btnMoyenne = new Button("Moyenne");
lMoy = new Label("Valeur");
```

Points clés

- `setToggleGroup()` → garantit qu'un seul `RadioButton` est sélectionné par groupe.
- `setSelected(true)` → sélectionne le coefficient 1 par défaut.
- `setValue()` → valeur initiale de la `ComboBox`.

Les Layouts en JavaFX

Qu'est-ce qu'un Layout ?

Un **layout** (ou *pane*) est un conteneur qui organise automatiquement la position de ses enfants (`Node`).

`FlowPane` Place les éléments les uns à côté des autres. Si la fenêtre est trop petite, ils passent à la ligne suivante.

`VBox` Empile les éléments **verticalement** (un par ligne).

`HBox` Place les éléments **horizontalement** (côte à côte).

`GridPane` Grille avec lignes et colonnes (comme un tableau).

addWidgetsToSceneV1() – FlowPane + VBox

```
private Scene addWidgetsToSceneV1() {
    // Un FlowPane par ligne de matière
    FlowPane fpAnglais = new FlowPane(5, 5); // (hgap, vgap)
    fpAnglais.getChildren().addAll(
        new Label("Anglais"), tfAnglais, rbAng1, rbAng2, rbAng3
    );
    // ... idem pour chaque matière

    FlowPane fpOption = new FlowPane(5, 5);
    fpOption.getChildren().addAll(
        listOption, tfOption, new Label("Optionnel")
    );

    FlowPane fpMoyenne = new FlowPane(5, 5);
    fpMoyenne.getChildren().addAll(btnMoyenne, lMoy);

    // VBox empile toutes les lignes verticalement
    VBox vbox = new VBox(10);        // espacement 10px
    vbox.setPadding(new Insets(10)); // marge intérieure
    vbox.getChildren().addAll(
        fpAnglais, fpMaths, fpInfo, fpGeo,
        fpOption, cbCoef, fpMoyenne
    );
    return new Scene(new FlowPane(vbox), 500, 300);
}
```

Gestion du redimensionnement (V1)

Problème

Si on met le `VBox` directement comme racine de la `Scene`, les widgets s'étirent pour occuper toute la fenêtre lors du redimensionnement.

Solution

On encapsule le `VBox` dans un `FlowPane` racine :

$$\texttt{Scene} \rightarrow \texttt{FlowPane (racine)} \rightarrow \texttt{VBox} \rightarrow \text{lignes}$$

Le `FlowPane` ne force pas ses enfants à s'étirer $\Rightarrow$ les widgets gardent leur taille naturelle.

addWidgetsToSceneV2() – GridPane

Principe du GridPane

Organise les widgets dans une grille. On spécifie la **colonne** et la **ligne** pour chaque élément : `grid.add(node, col, row)`.

```
private Scene addWidgetsToSceneV2() {
    // HBox pour regrouper les RadioButtons
    HBox hbAngRB = new HBox(5, rbAng1, rbAng2, rbAng3);
    // ...

    GridPane grid = new GridPane();
    grid.setHgap(10); grid.setVgap(10);

    // col 0 = Label, col 1 = TextField, col 2 = RadioButtons
    grid.add(new Label("Anglais"),       0, 0);
    grid.add(tfAnglais,                  1, 0);
    grid.add(hbAngRB,                    2, 0);

    grid.add(new Label("Mathématiques"),0, 1);
    grid.add(tfMaths,                    1, 1);
    grid.add(hbMathsRB,                  2, 1);
    // ... lignes 2, 3, 4 pour Info, Géo, Option
```

Version 2 – Cadre et assemblage final

```
    // Cadre autour du GridPane (CSS inline)
    grid.setStyle(
        "-fx-border-color: black;"
      + "-fx-border-width: 1;"
      + "-fx-padding: 10;");

    HBox hbMoyenne = new HBox(10, btnMoyenne, lMoy);

    VBox vbox = new VBox(10);
    vbox.setPadding(new Insets(10));
    vbox.getChildren().addAll(grid, cbCoef, hbMoyenne);

    // FlowPane racine pour le redimensionnement
    FlowPane root = new FlowPane();
    root.getChildren().add(vbox);

    return new Scene(root, 500, 350);
}
```

Hiérarchie V2

Scene → FlowPane → VBox → {GridPane, CheckBox, HBox}

Assembler le tout dans start()

```
@Override
public void start(Stage primaryStage) {
    initWidgets();

    // Choisir la version de l'interface :
    // Scene scene = addWidgetsToSceneV1();
    Scene scene = addWidgetsToSceneV2();

    primaryStage.setTitle("Calcul de la moyenne");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Ordre d'appel

1. `initWidgets()` – crée tous les composants
2. `addWidgetsToSceneV1/V2()` – les place dans un layout et retourne une Scene
3. Configuration du Stage (titre, scène, affichage)

Qu'est-ce qu'un évènement ?

Définition

Un **évènement** est un signal généré par une action de l'utilisateur : clic sur un bouton, saisie clavier, déplacement de souris...

Comment réagir à un évènement ?

On associe un **gestionnaire d'évènement** (*event handler*) au widget :

1. Le widget écoute un type d'évènement (ex. `ActionEvent` pour un clic bouton).
2. Quand l'évènement se produit, le gestionnaire exécute du code.

En JavaFX, la manière la plus simple est d'utiliser :

```
bouton.setOnAction(e -> maMethode());
```

Associer l'action au bouton

On ajoute cette ligne à la fin de `initWidgets()` :

```
// Dans initWidgets(), après la création du bouton :
btnMoyenne.setOnAction(e -> calculerMoyenne());
```

Explication de la syntaxe

`setOnAction()` Méthode qui enregistre un gestionnaire pour l'évènement `ActionEvent` (clic).

`e ->` **Expression lambda** – syntaxe compacte pour définir une action. e est l'objet évènement (souvent inutilisé ici).

`calculerMoyenne()` La méthode qui sera appelée à chaque clic.

La méthode calculerMoyenne() – Récupération

```
private void calculerMoyenne() {
  try {
    // 1. Récupérer les notes depuis les TextFields
    double noteAng   = Double.parseDouble(tfAnglais.getText());
    double noteMaths = Double.parseDouble(tfMaths.getText());
    double noteInfo  = Double.parseDouble(tfInfo.getText());
    double noteGeo   = Double.parseDouble(tfGeo.getText());
```

Méthodes clés

`getText()` Récupère le contenu du `TextField` sous forme de `String`.

`Double.parseDouble()` Convertit un `String` en `double`. Lève une `NumberFormatException` si le texte n'est pas un nombre valide.

La méthode calculerMoyenne() – Calcul

```
if (cbCoef.isSelected()) {
    // Moyenne pondérée
    int cAng = getCoef(tgAnglais); // ...
    double somme = noteAng*cAng + noteMaths*cMaths
                 + noteInfo*cInfo + noteGeo*cGeo;
    double totalCoef = cAng + cMaths + cInfo + cGeo;

    if (!tfOption.getText().isEmpty()) {
        somme += Double.parseDouble(tfOption.getText());
        totalCoef += 1;
    }
    moyenne = somme / totalCoef;
} else {
    // Moyenne simple (tous les coefs = 1)
    double somme = noteAng + noteMaths + noteInfo + noteGeo;
    int nb = 4;
    if (!tfOption.getText().isEmpty()) {
        somme += Double.parseDouble(tfOption.getText());
        nb++;
    }
    moyenne = somme / nb;
}
```

Récupérer le coefficient et afficher le résultat

```
// Méthode utilitaire pour lire le coefficient sélectionné
private int getCoef(ToggleGroup tg) {
    RadioButton selected = (RadioButton) tg.getSelectedToggle();
    return Integer.parseInt(selected.getText());
}
```

```
    // Affichage du résultat
    lMoy.setText(String.format("%.2f", moyenne));

  } catch (NumberFormatException ex) {
      lMoy.setText("Erreur de saisie");
  }
}
```

Points clés

- `getSelectedToggle()` retourne le `RadioButton` sélectionné dans le groupe.
- Le `try/catch` capture les erreurs si l'utilisateur entre du texte non numérique.
- `String.format("%.2f", ...)` formate le résultat à 2 décimales.

Récapitulatif – Ce que nous avons appris

1. **Structure d'une application JavaFX** : `Application` → `start()` → `Stage/Scene/Node`.
2. **Widgets** : `TextField`, `RadioButton`, `ToggleGroup`, `ComboBox`, `CheckBox`, `Button`, `Label`.
3. **Layouts** : `FlowPane`, `VBox`, `HBox`, `GridPane` – chacun organise les éléments différemment.
4. **Séparation des responsabilités** :
   - `initWidgets()` – création uniquement
   - `addWidgetsToSceneV1/V2()` – placement uniquement
5. **Gestion d'évènements** : `setOnAction()` avec expression lambda pour réagir aux clics.
6. **Redimensionnement** : encapsuler dans un `FlowPane` racine.

TP n°2 – Gestion des évènements en JavaFX

Objectif

Reprendre l'application du TP1 et y ajouter une **gestion complète des évènements**, en structurant le code avec des classes dédiées.

Ce que vous allez faire aujourd'hui :

1. **Classe** `ControlBouton` : créer une classe externe qui implémente `EventHandler<ActionEvent>` pour gérer le clic sur « Moyenne » (avec validation des notes, gestion des erreurs via `Alert`, et calcul complet).
2. **Méthode** `addListeners()` : associer le contrôleur au bouton de manière structurée.
3. **Modification de la fenêtre** : ajouter une icône, une position initiale, rendre la fenêtre non redimensionnable.
4. **Menu** `Options` : créer un `MenuBar` avec des items « Version 1 », « Version 2 » et un sous-menu « Aide ». Gérer le changement dynamique d'interface.

TP n°2 – Concepts clés à découvrir

Évènements & Contrôleurs

- Interface `EventHandler<ActionEvent>`
- Méthode `handle(ActionEvent e)`
- `getSource()` pour identifier l'élément cliqué
- Fenêtres de dialogue (`Alert`)

Techniques avancées

- Classes internes anonymes (section 5)
- `EventFilter` vs `EventHandler` (section 6)
- Changement dynamique de `Scene`
- Menus et sous-menus

Rappel

L'algorithme de calcul de la moyenne dans le TP2 est plus complet : il faut vérifier que les notes sont entre 0 et 20, gérer les champs vides, et afficher des messages d'erreur dans des fenêtres `Alert` modales.

Bon courage pour le TP2 !