# TP 6 : Interfaces graphiques et évènements avec JavaFX

**Matière :** ATELIER PROGRAMMATION OBJET AVANCEE | **Niveau :** DSI 2

**Enseignants : Equipe pédagogique**

1. Créer une application JavaFX "GestionPersonnes" permettant d'ajouter, modifier, supprimer une personne et d'afficher la liste des personnes.

2. En utilisant "Bouton Droit" -> Refractor -> Rename, renommer les 3 fichiers :
   - HelloApplication.java en PersonneApplication.java
   - HelloController.java en PersonneController.java
   - hello-view.fxml en personne-view.fxml

3. Créer un dossier model dans lequel déclarer une classe Personne

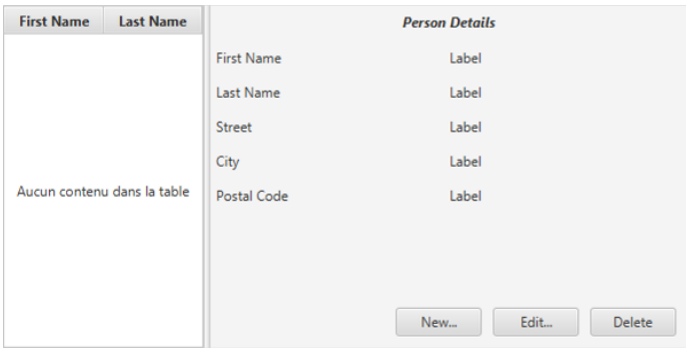
```
public class Personne {

    private final StringProperty firstName;
    private final StringProperty lastName;
    private final StringProperty street;
    private final IntegerProperty postalCode;
    private final StringProperty city;

    public Personne(){
        this.firstName = new SimpleStringProperty("") ;
        this.lastName = new SimpleStringProperty("");

        // Some initial dummy data, just for convenient testing.
        this.street = new SimpleStringProperty("");
        this.postalCode = new SimpleIntegerProperty(0);
        this.city = new SimpleStringProperty("");
    }
    public Personne(String firstName, String lastName, String street, Integer
postalCode, String city) {
        this.firstName = new SimpleStringProperty(firstName);
        this.lastName = new SimpleStringProperty(lastName);

        // Some initial dummy data, just for convenient testing.
        this.street = new SimpleStringProperty(street);
        this.postalCode = new SimpleIntegerProperty(postalCode);
        this.city = new SimpleStringProperty(city);
    }

    public String getFirstName() {
        return firstName.get();
    }

    public void setFirstName(String firstName) {
        this.firstName.set(firstName);
    }
```

```
    public StringProperty firstNameProperty() {
        return firstName;
    }

    public String getLastName() {
        return lastName.get();
    }

    public void setLastName(String lastName) {
        this.lastName.set(lastName);
    }

    public StringProperty lastNameProperty() {
        return lastName;
    }

    public String getStreet() {
        return street.get();
    }

    public void setStreet(String street) {
        this.street.set(street);
    }

    public StringProperty streetProperty() {
        return street;
    }

    public int getPostalCode() {
        return postalCode.get();
    }

    public void setPostalCode(int postalCode) {
        this.postalCode.set(postalCode);
    }

    public IntegerProperty postalCodeProperty() {
        return postalCode;
    }

    public String getCity() {
        return city.get();
    }

    public void setCity(String city) {
        this.city.set(city);
    }

    public StringProperty cityProperty() {
        return city;
    }
}
```

4. Dans le fichier "module-info.java", exporter le package "model" et permettre à javafx d'y accéder

```
module ... {
...
    exports model;
    opens model to javafx.fxml;
}
```

5. Préparer l'interface graphique de "personne-view.fxml"





6. Ouvrir le fichier "PersonneApplication.java" puis :
   a. Déclarer un objet de type Stage et implémenter son getter

```
public class PersonneApplication extends Application {
    private Stage primaryStage;

    public Stage getPrimaryStage() {
        return primaryStage;
    }
}
```

   b. Modifier la méthode start()

```
public void start(Stage primaryStage) throws IOException {
    this.primaryStage = primaryStage;
    FXMLLoader fxmlLoader = new FXMLLoader();
    PersonneController controller_person = new PersonneController();

    fxmlLoader.setController(controller_person);
    fxmlLoader.setLocation(PersonneApplication.class.getResource("personne-
view.fxml"));
    //FXMLLoader fxmlLoader = new
FXMLLoader(MainApplication.class.getResource("personne-view.fxml"));
```

```
    Scene scene = new Scene(fxmlLoader.load(), 600, 350);
    primaryStage.setTitle("Gestion des adresses");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

7. Ouvrir le fichier "PersonneController" puis :
   a. Effacer tout son contenu

```
public class PersonneController {

}
```

   b. Ajouter à "PersonneController.java" l'implémentation de l'interface "Initializable", déclarer et appeler les méthodes initGraphique() et ecouteurs(); :

```
public class PersonneController implements Initializable {

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        initGraphique();
        ecouteurs();
    }

    private void initGraphique() {
    }

    private void ecouteurs() {
    }
}
```

   c. Déclarer les contrôles de l'interface

```
@FXML
private TableView<Personne> personTable;
@FXML
private TableColumn<Personne, String> firstNameColumn;
@FXML
private TableColumn<Personne, String> lastNameColumn;

@FXML
private Label firstNameLabel;
@FXML
private Label lastNameLabel;
@FXML
private Label streetLabel;
@FXML
private Label postalCodeLabel;
@FXML
private Label cityLabel;
```

   d. Déclarer la méthode showPersonDetails(Personne p) permettant d'afficher les détails d'une personne dans les labels correspondant

```
private void showPersonDetails(Personne p) {
    if (p != null) {
        // Fill the labels with info from the person object.
        firstNameLabel.setText(p.getFirstName());
        lastNameLabel.setText(p.getLastName());
        streetLabel.setText(p.getStreet());
```

```
        postalCodeLabel.setText(Integer.toString(p.getPostalCode()));
        cityLabel.setText(p.getCity());

    } else {
        // Person is null, remove all the text.
        firstNameLabel.setText(" ");
        lastNameLabel.setText(" ");
        streetLabel.setText(" ");
        postalCodeLabel.setText(" ");
        cityLabel.setText(" ");
    }
}
```

e. Déclarer également un ObservableList<Personne> et développer la méthode initGraphique() pour le remplir par des instances de la classe personnes

```
private ObservableList<Personne> people =
FXCollections.observableArrayList();

private void initGraphique() {
    Personne person1 = new Personne("John", "Doe", "123 Main St", 12345,
"Anytown");
    Personne person2 = new Personne("Jane", "Smith", "456 Oak St", 54321,
"Othertown");
    Personne person3 = new Personne("Alice", "Johnson", "789 Elm St",
67890, "AnotherTown");
    Personne person4 = new Personne("Bob", "Brown", "101 Maple St", 10101,
"Somewhere");
    Personne person5 = new Personne("Eve", "Williams", "111 Pine St",
11111, "Nowhere");
    people.addAll(person1, person2, person3, person4, person5);

    firstNameColumn.setCellValueFactory(new
PropertyValueFactory<>("firstName"));
    lastNameColumn.setCellValueFactory(new
PropertyValueFactory<>("lastName"));

    personTable.getItems().addAll(people);
    personTable.getSelectionModel().select(person1);

    // Clear person details.
    showPersonDetails(person1);
}
```

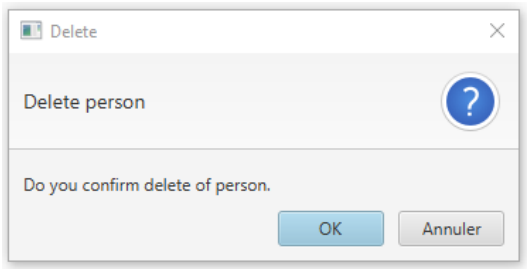
f. Ajouter ce code à "ecouteurs()" :

```
private void ecouteurs() {
  // Listen for selection changes and show the person details when changed.
personTable.getSelectionModel().selectedItemProperty().addListener(
(observable, oldValue, newValue) -> showPersonDetails(newValue));
}
```

g. Définir un objet mainApp de type PersonneApplication et implémenter son setter

```
// Reference to the main application.
private PersonneApplication mainApp;

public void setMainApp(PersonneApplication mainApp) {
    this.mainApp = mainApp;
}
```

h. Ajouter un écouteur au bouton « Delete » pour qu'il appelle la méthode handleDeletePerson() permettant de supprimer une personne de la liste après confirmation




```
@FXML
private void handleDeletePerson() {
    int selectedIndex = personTable.getSelectionModel().getSelectedIndex();
    if (selectedIndex >= 0) {
        // Nothing selected.
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.initOwner(mainApp.getPrimaryStage());
        alert.setTitle("Delete");
        alert.setHeaderText("Delete person");
        alert.setContentText("Do you confirm delete of person.");

        alert.showAndWait();
        if (alert.getResult() == ButtonType.OK) {
            personTable.getItems().remove(selectedIndex);
        }

    } else {
        // Nothing selected.
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.initOwner(mainApp.getPrimaryStage());
        alert.setTitle("No Selection");
        alert.setHeaderText("No Person Selected");
        alert.setContentText("Please select a person in the table.");

        alert.showAndWait();
    }
}
```

8. Ouvrir le fichier "PersonneApplication.java" et modifier la méthode start()

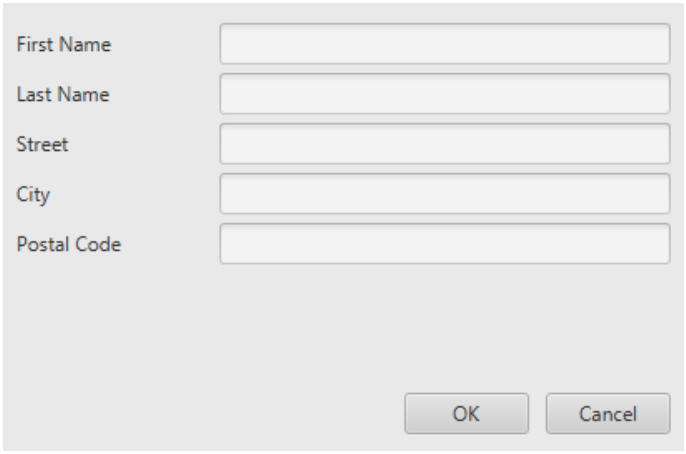
```
public void start(Stage primaryStage) throws IOException {
...
    PersonneController controller_person = new PersonneController();
    controller_person.setMainApp(this);
...
}
```

9. Ouvrir le fichier "personne-view.fxml " et supprimer l'appel du contrôleur de la racine du graphe de scène

```
<AnchorPane xmlns:fx="http://javafx.com/fxml"
fx:controller="com.samihadhri.gestpersonnes.PersonneController">
```

10. Lancer l'application et tester le bouton de suppression

11. Créer un nouveau fichier d'interface "personne-edit-view.fxml" et préparer son interface graphique





12. Créer un fichier "PersonneEditController.java"
    a. Déclarer les contrôles de l'interface

```
public class PersonneEditController {

    @FXML
    private TextField firstNameField;
    @FXML
    private TextField lastNameField;
    @FXML
    private TextField streetField;
    @FXML
    private TextField postalCodeField;
    @FXML
    private TextField cityField;


    private Stage dialogStage;
    private Personne person;
    private boolean okClicked = false;
}
```

    b. Déclarer un objet de type Stage et définir son setteur

```
...
    private Stage dialogStage;

    public void setDialogStage(Stage dialogStage) {
        this.dialogStage = dialogStage;
    }
...
```

    c. Déclarer un objet de type Personne et définir son setteur

```
...
    private Personne person;
```

```
    public void setPerson(Personne person) {
        this.person = person;

        firstNameField.setText(person.getFirstName());
        lastNameField.setText(person.getLastName());
        streetField.setText(person.getStreet());
        postalCodeField.setText(Integer.toString(person.getPostalCode()));
        cityField.setText(person.getCity());
    }
...
```

d. Déclarer un objet de type booléen et définir son getteur

```
...
    private boolean okClicked = false;

    public boolean isOkClicked() {
        return okClicked;
    }
...
```

e. Déclarer une méthode isInputValid() qui retourne true si tous les champs sont valides et affiche une boite de dialogue et retourne false si un ou plusieurs champs sont invalides


```
private boolean isInputValid() {
    String errorMessage = "";

    if (firstNameField.getText() == null || firstNameField.getText().length()
== 0) {
        errorMessage += "No valid first name!\n";
    }
    if (lastNameField.getText() == null || lastNameField.getText().length()
== 0) {
        errorMessage += "No valid last name!\n";
    }
    if (streetField.getText() == null || streetField.getText().length() == 0)
{
        errorMessage += "No valid street!\n";
    }

    if (postalCodeField.getText() == null ||
postalCodeField.getText().length() == 0) {
        errorMessage += "No valid postal code!\n";
    } else {
        // try to parse the postal code into an int.
        try {
            Integer.parseInt(postalCodeField.getText());
        } catch (NumberFormatException e) {
```

```
            errorMessage += "No valid postal code (must be an integer)!\n";
        }
    }

    if (cityField.getText() == null || cityField.getText().length() == 0) {
        errorMessage += "No valid city!\n";
    }

    if (errorMessage.length() == 0) {
        return true;
    } else {
        // Show the error message.
        Alert alert = new Alert(AlertType.ERROR);
        alert.initOwner(dialogStage);
        alert.setTitle("Invalid Fields");
        alert.setHeaderText("Please correct invalid fields");
        alert.setContentText(errorMessage);

        alert.showAndWait();

        return false;
    }
}
```

f. Ajouter un écouteur au bouton « OK » pour qu'il appelle la méthode handleOk () permettant de vérifie si les données saisies sont valides et met à jour les champs de l'objet person avec les valeurs saisies dans les champs du formulaire puis elle ferme la fenêtre de dialogue "dialogStage"

```
private void handleOk() {
    if (isInputValid()) {
        person.setFirstName(firstNameField.getText());
        person.setLastName(lastNameField.getText());
        person.setStreet(streetField.getText());
        person.setPostalCode(Integer.parseInt(postalCodeField.getText()));
        person.setCity(cityField.getText());

        okClicked = true;
        dialogStage.close();
    }
}
```

g. Ajouter un écouteur au bouton « Cancel » pour qu'il appelle la méthode handleCancel() permettant de fermer la fenêtre de dialogue "dialogStage"

```
private void handleCancel() {
    dialogStage.close();
}
```

13. Ajouter dans le fichier "PersonneApplication.java" la méthode showPersonEditDialog(Personne person) qui charge et affiche une fenêtre de dialogue modale pour permettre à l'utilisateur de modifier les détails d'une personne, renvoyant vrai si les modifications sont confirmées et faux sinon

```
public boolean showPersonEditDialog(Personne person) {
    try {
        // Load the fxml file and create a new stage for the popup dialog.
        FXMLLoader loader = new FXMLLoader();
        PersonneEditController controller_person = new
PersonneEditController();
```

```
        loader.setController(controller_person);
        loader.setLocation(PersonneApplication.class.getResource("personne-
edit-view.fxml"));
        AnchorPane page = (AnchorPane) loader.load();

        // Create the dialog Stage.
        Stage dialogStage = new Stage();
        dialogStage.setTitle("Edit Person");
        dialogStage.initModality(Modality.WINDOW_MODAL);
        dialogStage.initOwner(primaryStage);
        Scene scene = new Scene(page);
        dialogStage.setScene(scene);

        // Set the person into the controller.
        PersonneEditController controller = loader.getController();
        controller.setDialogStage(dialogStage);
        controller.setPerson(person);

        // Show the dialog and wait until the user closes it
        dialogStage.showAndWait();
        return controller.isOkClicked();
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
```

14. Dans le fichier "PersonneController.java" :
    a. Ajouter un écouteur au bouton « New » pour qu'il appelle la méthode handleNewPerson() permettant de créer d'abord une nouvelle instance de la classe Personne, puis affiche une fenêtre de dialogue modale pour permettre à l'utilisateur de saisir les détails de la nouvelle personne. Si l'utilisateur clique sur "OK" dans la fenêtre de dialogue pour confirmer l'ajout, la nouvelle personne est ajoutée à la liste affichée dans la table des personnes. Enfin, la table est actualisée pour refléter les changements

```
@FXML
private void handleNewPerson() {
    Personne tempPerson = new Personne();
    boolean okClicked = mainApp.showPersonEditDialog(tempPerson);
    if (okClicked) {
        System.out.print(people.size());
        personTable.getItems().add(tempPerson);
        // people.add(tempPerson);
        //  personTable.getColumns().clear();

        personTable.refresh();
    }
}
```

    b. Ajouter un écouteur au bouton « Edit » pour qu'il appelle la méthode handleCancel() permettant de gérer l'édition des détails d'une personne sélectionnée dans la table en affichant une fenêtre de dialogue modale pour effectuer les modifications, actualisant ensuite la table si les modifications sont confirmées, sinon affichant une alerte si aucune personne n'est sélectionnée

```
@FXML
private void handleEditPerson() {
```

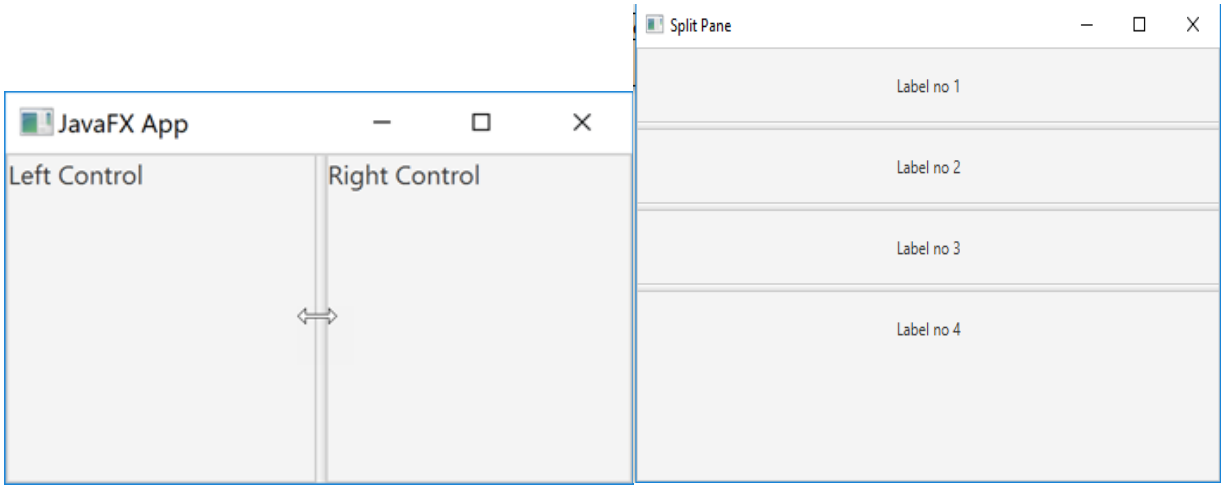
```
    Personne selectedPerson =
personTable.getSelectionModel().getSelectedItem();
    if (selectedPerson != null) {
        boolean okClicked = mainApp.showPersonEditDialog(selectedPerson);
        if (okClicked) {
            showPersonDetails(selectedPerson);
        }

    } else {
        // Nothing selected.
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.initOwner(mainApp.getPrimaryStage());
        alert.setTitle("No Selection");
        alert.setHeaderText("No Person Selected");
        alert.setContentText("Please select a person in the table.");

        alert.showAndWait();
    }
}
```

# Annexe

## JavaFX SplitPane

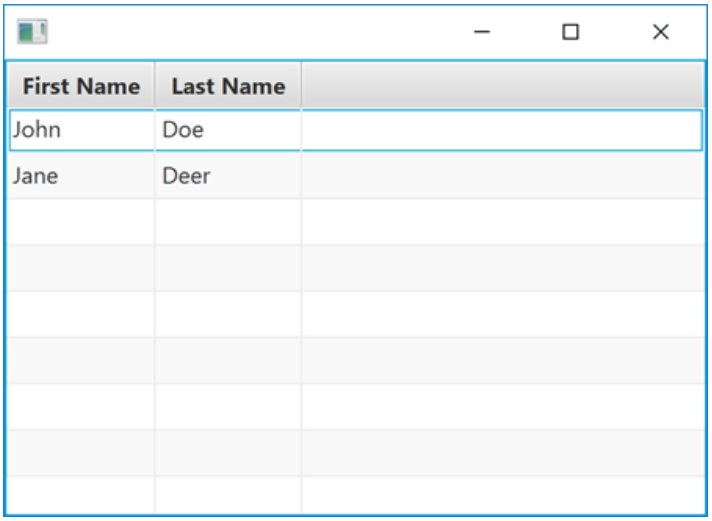
Le contrôle JavaFX SplitPane est un conteneur qui peut contenir plusieurs autres composants. Le SplitPane est divisé en 2 zones ou plus séparé par un séparateur. L'utilisateur peut déplacer le séparateur pour définir la quantité d'espace allouée à chaque contrôle. On peut aussi changer d'orientation (horizontale ou verticale).

Voici une capture d'écran d'un JavaFX SplitPane :




## JavaFx tableview

JavaFX TableView permet d'afficher des vues de tableau dans les applications JavaFX. Le JavaFX TableView est représenté par la classe javafx.scene.control.TableView . Voici une capture d'écran d'un JavaFX TableView :



**Classes liées à TableView**

La classe JavaFX TableView utilise un ensemble de classes associées pour faire son travail. Ces classes sont :

- TableColumn
- TableRow
- TableCell
- TablePosition
- TableViewFocusModel
- TableViewSelectionModel

Un TableView est composé d'un certain nombre d'instances de TableColumn. Chaque TableColumn d'un tableau est responsable de l'affichage (et de la modification) du contenu de cette colonne. En plus d'être responsable de l'affichage et de la modification des données d'une seule colonne, une TableColumn contient également les propriétés nécessaires pour :

- Être redimensionné (en utilisant les propriétés minWidth/prefWidth/maxWidth et width)
- Afficher le texte de l'en-tête
- Afficher toutes les colonnes imbriquées qu'il peut contenir
- Avoir un menu contextuel lorsque l'utilisateur clique avec le bouton droit sur la zone d'en-tête de colonne
- Faire en sorte que le contenu de la table soit trié (à l'aide du comparator, sortable and sortType)

Lors de la création d'une instance TableColumn, les deux propriétés les plus importantes à définir sont peut-être le texte de la colonne (ce qu'il faut afficher dans la zone d'en-tête de colonne) et la fabrique de valeurs de cellule de colonne (qui est utilisée pour remplir les cellules individuelles de la colonne). Ceci peut être réalisé en utilisant une variante du code suivant :

```
ObservableList<Person> data = ...
TableView<Person> tableView = new TableView<Person>(data);
TableColumn<Person, String> firstNameCol = new TableColumn<Person, String>("First
Name");
 firstNameCol.setCellValueFactory(new Callback<CellDataFeatures<Person, String>,
ObservableValue<String>>()
 {
   public ObservableValue<String> call (CellDataFeatures < Person, String > p){
      // p.getValue() returns the Person instance for a particular TableView row
      return p.getValue().firstNameProperty();
   }
 });
tableView.getColumns().add(firstNameCol);
```