

2022

ARCHITECTURE DES INTERFACES HUMAIN-MACHINE

JAVAFX

P. STUDER

ENSISA

EXEMPLE 3

1. CONTENEURS, LISTE

Nous allons créer une application qui présente une liste d'oiseaux et des informations de détail sur l'oiseau sélectionné.

Des oiseaux

Editor Supprimer Filtre

Mésange charbonnière Parus major
Pinson des arbres <i>Fringilla coelebs</i>
Merle noir Turdus merula
Pouillot véloce Phylloscopus collybita
Mésange bleue Cyanistes caeruleus
Rougegorge familier Erithacus rubecula
Fauvette à tête noire Sylvia atricapilla
Pouillot fitis Phylloscopus trochilus
Grive musicienne Turdus philomelos
Bruant jaune Emberiza citrinella
Troglodyte mignon Troglodytes troglodytes
Bruant proyer

Pinson des arbres
Fringilla coelebs

Fringillidae
Fringilla
Coelebs



Le Pinson des arbres est un passereau commun et facile à reconnaître. Le mâle adulte nuptial est assez bariolé. Le manteau et le haut du dos sont d'un brun-marron chaud. La tête est d'un gris-ardoise bleuté, excepté le front qui est noir et l'ensemble "lores, joues et couvertures auriculaires" qui est châtain. Le bec est gris bleuté. Deux larges barres blanches séparées de noir sur les couvertures alaires sont diagnostiques et très visibles, au posé comme en vol. Celle des grandes couvertures tend souvent vers le jaune clair. Les rémiges sombres sont ourlées de jaune. Le croupion et les sus-caudales sont olive. La queue sombre montre du blanc aux deux paires de rectrices externes. Les parties inférieures sont d'un rose vineux prononcé, avec parfois une nuance roussâtre. Le bas ventre et les sous-caudales sont blancs. Les pattes sont rosâtres. En hiver, les couleurs sont atténues et c'est l'usure du plumage qui fera apparaître les belles couleurs sous-jacentes pour la reproduction.

1.1. CRÉATION DU PROJET

✍ Dans IntelliJ, créez un nouveau projet JavaFX nommé **birds**, group : **ensisa**

✍ Editez **pom.xml**

Changez la version de JavaFX en 21 (ou +) si le document comporte une version antérieure :

```
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>21</version>
</dependency>
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>21</version>
</dependency>
```

Clic-droit sur **pom.xml** > **Maven** > **Reload project**

✍ Editez **module-info.java**

```
module ensisa.birds {
    requires javafx.controls;
    requires javafx.fxml;

    opens ensisa.birds to javafx.fxml;
    exports ensisa.birds;
}
```

✍ Renommez la classe **HelloApplication** en **BirdApplication** en utilisant la fonction de renommage d'IntelliJ

👉 Dans **resources.ensisa.birds**, créez un fichier fxml nommé **main-view.fxml**

Changez le nom du contrôleur en : [ensisa.birds.MainController](#)

👉 Dans **java.ensisa.birds**, créez la classe [ensisa.birds.MainController](#)

👉 Dans la méthode [start](#) de [BirdApplication](#), remplacez le nom du fichier fxml à charger et modifiez quelques propriétés :

```
public void start(Stage stage) throws IOException {  
    FXMLLoader fxmlLoader = new FXMLLoader(  
        BirdApplication.class.getResource("main-view.fxml"));  
    Scene scene = new Scene(fxmlLoader.load(), 800, 600);  
    stage.setTitle("Des oiseaux");  
    stage.setScene(scene);  
    stage.show();  
}
```

👉 Supprimez les fichiers **hello-view.fxml** et **HelloController.java**

👉 Lancez l'application pour vérifier qu'il n'y a pas d'erreurs.

1.2.LE MODÈLE

Dans un premier temps, nous allons créer un modèle passif pour décrire les oiseaux.

👉 Dans **resources.ensisa.birds**, créez un dossier **assets**

👉 Dans **assets**, glissez le fichier **Birds.json** qui se trouve dans le dossier "BirdsData" téléchargé depuis Moodle.

👉 Dans **assets**, glissez le dossier **images** avec son contenu qui se trouve dans le dossier "BirdsData" téléchargé depuis Moodle.

👉 Dans **java**, créez un paquetage java nommé [ensisa.birds.model](#)

👉 Dans ce paquetage, glissez le fichier **Bird.java** qui se trouve dans le dossier "BirdsData/Passif"

Regardez le contenu du fichier.

Nous allons ensuite modéliser une base d'oiseaux. La base sera peuplée à partir du fichier **Birds.json**. Pour lire le format json, nous utilisons le paquetage **Jackson**.

👉 Ajoutez une dépendance dans le fichier **pom.xml** , puis **Maven > Reload project** :

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.13.4.1</version>
</dependency>
```

👉 Modifiez les informations sur les modules dans le fichier **module-info.java** :

```
module ensisa.birds {
    requires transitive javafx.controls;
    requires javafx.fxml;

    requires com.fasterxml.jackson.databind;

    opens ensisa.birds to javafx.fxml, com.fasterxml.jackson.databind;
    exports ensisa.birds;
    exports ensisa.birds.model;
}
```

👉 Dans le paquetage **ensisa.birds.model**, glissez le fichier **BirdRepository.java** qui se trouve dans le dossier "BirdsData/Passif"

Regardez le contenu du fichier.

👉 Modifiez la classe **MainController** :

```
public class MainController {
```

```

private BirdRepository repository;

public MainController() {
    repository = new BirdRepository();
    repository.load();
}

}

```

1.3. EBAUCHE DE LA VUE DE DÉTAIL SUR UN OISEAU

 Ouvrez le fichier `main-view.fxml` à l'aide de **Scene Builder**

Le fichier contient un conteneur `AnchorPane` par défaut. `AnchorPane` permet de disposer ses composants enfants en absolu par rapport à ses bords.

Supprimez `AnchorPane`.

Mettez un composant `VBox` comme racine de la scène. `VBox` dispose ses enfants en colonne. Mettez `Spacing` à 5.

En supprimant le panneau racine, la classe du contrôleur a été perdue. Dans le panneau **Controller** en bas à gauche, entrez la classe du contrôleur : `ensisa.birds.MainController`

Imbriquez un deuxième `VBox` dans le premier. Ce composant contiendra le nom commun et le nom en latin de l'oiseau.

Imbriquez un `Label` dans cette `VBox`. Changez la police pour Arial, 24 pt, bold (par exemple). Affectez un identifiant (`fx:id`) : `commonNameLabel`

Glissez un deuxième `Label` sous le premier (dans la même `VBox`). Changez la police pour Arial, 13 pt, regular. Changez la couleur du texte pour #4d4d4d. Affectez un identifiant (`fx:id`) : `latin nameLabel`

Sélectionnez le `VBox` imbriqué. Mettez `Spacing` à 5.

Et on met un peu d'espace autour des informations. Sélectionnez le `VBox` racine, et mettez `Padding` à 20, 20, 20, 20.

Dans le menu **Preview** de **Scene Builder**, vous pouvez prévisualiser l'interface.

Sauvegardez le fichier **main-view.fxml** ; son contenu devrait être :

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" spacing="5.0"
xmlns="http://javafx.com/javafx/22" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ensisa.birds.MainController">
    <children>
        <VBox prefHeight="200.0" prefWidth="100.0" spacing="5.0">
            <children>
                <Label fx:id="commonNameLabel" text="Label">
                    <font>
                        <Font name="Arial Bold" size="24.0" />
                    </font>
                </Label>
                <Label fx:id="latinNameLabel" text="Label" textFill="#4d4d4d">
                    <font>
                        <Font name="Arial" size="13.0" />
                    </font>
                </Label>
            </children>
        </VBox>
    </children>
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
    </padding>
</VBox>
```

👉 Lancez l'application pour voir l'interface. Nous allons remplacer les textes "Label" par des informations issues d'un oiseau.

Lorsque JavaFX charge un fichier fxml, il réalise cette séquence d'opérations :

- Chargement du fichier fxml
- Appel du constructeur du contrôleur
- Injection des variables d'instance
- Appel à une méthode nommée `initialize()` dans le contrôleur

Pour référencer les labels depuis le contrôleur, il faut définir des variables d'instances correspondantes nommées avec les noms associés à `fx:id`. Ces variables sont initialisées lors de la phase d'injection. Pour pouvoir réaliser l'injection, ces variables doivent être annotées par `@FXML`.

Pourquoi existe-t-il une méthode `initialize` en plus du constructeur ? Dans le constructeur, l'injection n'est pas faite, donc les variables sont nulles. Dans `initialize`, l'injection a été faite, les variables réfèrent donc réellement les composants.

 Ajoutez les variables correspondantes aux labels :

```
public class MainController {
    private BirdRepository repository;

    @FXML
    private Label commonNameLabel;
    @FXML
    private Label latinNameLabel;
    ...
}
```

 Déclarez une variable `currentBird` qui référence l'oiseau couramment visualisé et écrivez la méthode `initialize()` dans laquelle on va modifier les textes des labels :

```
public class MainController {
    private BirdRepository repository;

    @FXML
    private Label commonNameLabel;
```

```

@FXML
private Label latinNameLabel;

public Bird currentBird;

public MainController(){
    repository = new BirdRepository();
    repository.Load();

    currentBird = repository.birds.get(0);
}

public void initialize() {
    commonNameLabel.setText(currentBird.getCommonName());
    latinNameLabel.setText(currentBird.getLatinName());
}

}

```

👉 Lancez l'application. Les noms correspondent au premier oiseau figurant dans la base.

1.4. MODÈLE ACTIF

Nous utilisons un modèle qui utilise la notion de propriété.

👉 Supprimez le fichier **Bird.java**

👉 Dans le dossier **model**, glissez le fichier **Bird.java** qui se trouve dans le dossier "BirdsData/Actif"

Remarquez que la classe comporte des propriétés.

👉 Supprimez le fichier **BirdRepository.java**

👉 Glissez le fichier **BirdRepository.java** qui se trouve dans le dossier "BirdsData/Actif"

Remarque que la liste des oiseaux est une collection observable.

✍ Dans la classe `MainController`, supprimez des instructions pour revenir à ce code :

```
public MainController() {
    repository = new BirdRepository();
    repository.Load();

    currentBird = repository.birds.get(0);
}

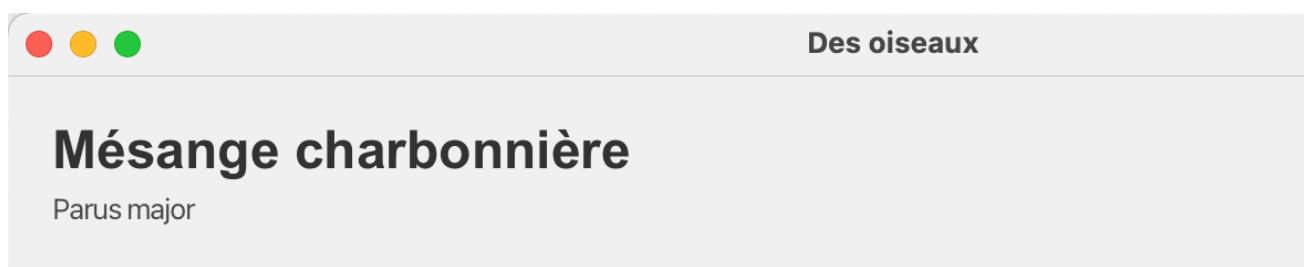
public void initialize() {

}
```

✍ Définissez les liaisons de données dans la méthode `initialize` :

```
public void initialize() {
    commonNameLabel.textProperty().bind(currentBird.commonNameProperty());
    latinNameLabel.textProperty().bind(currentBird.latinNameProperty());
}
```

✍ Lancez l'application. Les noms s'affichent comme auparavant si les liaisons de données ont été correctement écrites.



1.5. LA VUE DE DÉTAIL DE L'OISEAU, SUITE...

Sous les 2 textes de titre, nous allons afficher à gauche la famille, le genre, l'espèce de l'oiseau et à droite l'image de l'oiseau. Puis en-dessous, la description de l'oiseau.

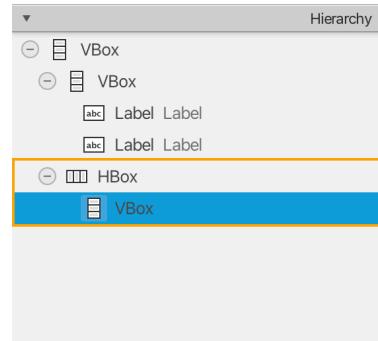
Editez `main-view.fxml` dans Scene Builder

Sélectionnez le `VBox` imbriqué (des titres) et dans le menu **Modify**, sélectionnez **Use Computed Sizes**. La taille du `VBox` est calculé à l'exécution.

Imbriquez un `HBox` dans la `VBox` racine (pas dans, mais sous la `VBox` des titres).

Imbriquez un `VBox` dans la `HBox`. Mettez **Pref Width** à 200 et **Pref Height** à `USE_COMPUTED_SIZE`. Mettez **Alignment** à `CENTER_LEFT`.

Dans la classe `MainController`, ajoutez ces variables d'instance :



```

@FXML
private Label familyLabel;
@FXML
private Label genusLabel;
@FXML
private Label specieLabel;
@FXML
private Label descriptionLabel;

```

Dans la `VBox` qui vient d'être créée, ajoutez 3 labels avec les `fx:id` : `familyLabel`, `genusLabel` et `specieLabel`

```

<VBox spacing="10.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="ensisa.birds.MainController">
    <children>
        <VBox spacing="5.0">
            <children>
                <Label fx:id="commonNameLabel" text="Label">

```

```

<font>
    <Font name="Arial Bold" size="24.0" />
</font>
</Label>
<Label fx:id="latinNameLabel" text="Label" textFill="#4d4d4d">
    <font>
        <Font name="Arial" size="13.0" />
    </font>
</Label>
</children>
</VBox>
<HBox prefHeight="100.0" prefWidth="200.0">
    <children>
        <VBox alignment="CENTER_LEFT" prefWidth="200.0">
            <children>
                <Label fx:id="familyLabel" text="Label"/>
                <Label fx:id="genusLabel" text="Label"/>
                <Label fx:id="specieLabel" text="Label"/>
            </children>
        </VBox>
    </children>
</HBox>
</children>
<padding>
    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
</padding>
</VBox>

```

 Dans la classe [MainController](#), établissez les liens :

```

public void initialize() {
    commonNameLabel.textProperty().bind(currentBird.commonNameProperty());
    latinNameLabel.textProperty().bind(currentBird.latinNameProperty());
    familyLabel.textProperty().bind(currentBird.familyProperty());
    genusLabel.textProperty().bind(currentBird.genusProperty());
    specieLabel.textProperty().bind(currentBird.specieProperty());
}

```

👉 Imbriquez un **Label** dans le **VBox** racine avec **wrap text** coché, **Text Alignment** : justifié et **fx:id** : **descriptionLabel**.

👉 Dans la classe **MainController**, établissez les liens :

```
descriptionLabel.textProperty().bind(currentBird.descriptionProperty());
```

👉 Lancez l'application.



1.6. STYLE

👉 Sélectionnez les labels et choisissez la police de caractères **Arial**

👉 Sélectionnez **italic** pour les 3 labels dans la **VBox**

Il reste à afficher l'image de l'oiseau.

1.7. LECTURE D'UNE IMAGE EN RESSOURCES

👉 Dans la classe **MainController** :

```
@FXML  
private ImageView birdImageView;
```

✍ Pour la **HBox**, mettez **Pref Height** à 300 et **Min height** à **USE_PREF_SIZE**

✍ Dans la **HBox**, glissez un composant **ImageView** avec **fx:id** : **birdImageView**, **fitHeight** à 300, **fitWidth** à 0 et **preserveRatio** coché.

✍ Dans la classe **Bird**, ajoutez une propriété pour récupérer une image JavaFX de l'oiseau :

```
@JsonIgnore  
private Property<Image> image = new SimpleObjectProperty<>(this, "image");
```

✍ Générez les accesseurs/mutateurs

```
public Image getImage(){  
    return image.getValue();  
}  
  
public Property<Image> imageProperty(){  
    return image;  
}  
  
public void setImage(Image image){  
    this.image.setValue(image);  
}
```

✍ Modifiez le constructeur :

Avec un écouteur d'événement :

```
public Bird(){  
    imagePath.addListener((v, oldValue, newValue) -> {  
        var url = getClass().getResource("/ensisa/birds/assets/images/" +  
            getImagePath().toExternalForm());  
        setImage(new Image(url.toString()));  
    });  
}
```

```
});  
}
```

Ou avec la méthode map :

```
public Bird(){  
    image.bind(imagePath.map(path -> {  
        var url = getClass().getResource("/ensis/birds/assets/images/" +  
            getImagePath()).toExternalForm();  
        return new Image(url.toString());  
    }));  
}
```

 Dans la classe [MainController](#), établissez les liens :

```
birdImageView.imageProperty().bind(currentBird.imageProperty());
```

Des oiseaux



Mésange charbonnière

Parus major

Paridae
Parus
Major

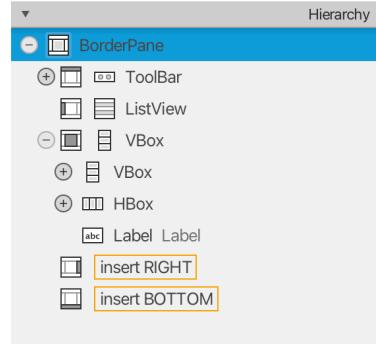


La Mésange charbonnière est une grande mésange, la plus grande de celles auxquelles nous sommes confrontés dans l'ouest du continent. Elle est remarquable par sa tête noire à larges joues blanches. L'œil très sombre est noyé dans ce noir. Le dessus du corps, manteau et dos, est verdâtre avec une zone plus claire sur la nuque. Les ailes et la queue sombres sont d'un gris nettement nuancé de bleu. Une barre alaire blanche se distingue sur les grandes couvertures alaires. Les parties inférieures sont jaunes mais s'éclaircissent vers la queue. L'arrière des flancs est gris. Le bas-ventre et les sous-caudales sont blancs avec un trait noir médian sur ces dernières. Un bandeau noir médio-ventral court de la gorge au ventre, large et d'un noir profond chez le mâle adulte, plus restreint et plus irrégulier chez la femelle.

 Lancez l'application.

1.8. AJOUT DE L'AFFICHAGE D'UNE LISTE D'OISEAUX

 Editez `main-view.fxml`. Imbriquez la vue précédente (le `VBox` racine) dans un composant `BorderPane` (clic droit > Wrap in... > BorderPane). Au centre se trouve la vue de l'oiseau. En haut, mettez une barre de commandes `ToolBar` et à gauche une liste `ListView`.



 Dans la classe `MainController`, déclarez une variable pour référencer la liste :

```
@FXML  
private ListView<Bird> birdListView;
```

Puis associez la liste du modèle au composant dans la méthode `initialize` :

```
birdListView.setItems(repository.birds);
```

 Dans `main-view.fxml`, associez `birdListView` à `fx:id` pour la liste, mettez Pref Height à `USE_COMPUTED_SIZE`

 Dans `main-view.fxml`, pour la `VBox` au centre du `BorderPane`, mettez Pref Height à `USE_COMPUTED_SIZE`

 Lancez l'application ; les cellules affichent la représentation textuelle des objets de type `Bird`.

Il faut définir l'apparence des cellules.

Chaque cellule est affichée par une instance de la classe `ListCell`. Cette classe comporte une propriété `Text` qui correspond à la valeur textuelle affichée par la cellule. Lorsqu'une cellule apparaît, la méthode `updateItem` est invoquée ; c'est donc cette méthode qu'il

faut redéfinir pour modifier la propriété `Text`. Pour générer des cellules d'apparence personnalisée, il faut associer à `ListView` une fabrique de cellules.

✍ Créez une classe `BirdCellFactory` :

```
public class BirdCellFactory implements Callback<ListView<Bird>, ListCell<Bird>>
{
    @Override
    public ListCell<Bird> call(ListView<Bird> param) {
        return new ListCell<>(){
            public void updateItem(Bird bird, boolean empty) {
                super.updateItem(bird, empty);
                if (empty || bird == null) {
                    setText(null);
                } else {
                    setText(bird.getCommonName());
                }
            };
        };
    }
}
```

✍ Dans la classe `MainController` :

```
birdListView.setCellFactory(new BirdCellFactory());
birdListView.setItems(repository.birds);
```

✍ Lancez l'application

Nous allons personnaliser la cellule avec un fichier fxml.

✍ Dans la classe `ListCell` de `BirdCellFactory`, ajoutez des variables pour référencer des composants fxml :

```
return new ListCell<>(){
```

```

@FXML
private VBox vBox;
@FXML
private Label commonNameLabel;
@FXML
private Label latinNameLabel;
private FXMLLoader loader;

```

Puis écrivez le code qui va lire le fichier fxml et mettre à jour les données :

```

public void updateItem(Bird bird, boolean empty) {
    super.updateItem(bird, empty);
    if (empty || bird == null) {
        setText(null);
        setGraphic(null);
    } else {
        if (loader == null) {
            loader = new FXMLLoader(getClass().getResource("bird-cell.fxml"));
            loader.setController(this);
            try {
                loader.load();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        commonNameLabel.textProperty().bind(
            bird.commonNameProperty());
        latinNameLabel.textProperty().bind(bird.latinNameProperty());
        setText(null);
        setGraphic(vBox);
    }
};

```

 Dans resources.ensisa.birds, créez un fichier bird-cell.fxml

```

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<VBox fx:id="vBox" alignment="CENTER_LEFT" minWidth="200.0" spacing="2.0"
xmlns="http://javafx.com/javafx/22" xmlns:fx="http://javafx.com/fxml/1">
<children>
    <Label fx:id="commonNameLabel" text="Label">
        <font>
            <Font size="14.0" />
        </font>
    </Label>
    <Label fx:id="latinNameLabel" text="Label" textFill="#545454" />
</children>
<padding>
    <Insets bottom="2.0" top="2.0" />
</padding>
</VBox>

```

 Lancez l'application

1.9. PRISE EN COMPTE DE LA SÉLECTION D'UN OISEAU

L'oiseau courant dépend de la sélection dans la liste. Il faut que l'on puisse observer cette propriété.

 Dans la classe **MainController**, remplacez la déclaration de currentBird par :

```
private final ObjectProperty<Bird> currentBird;
```

 Dans le constructeur, remplacez l'initialisation par :

```
currentBird = new SimpleObjectProperty<>(repository.birds.get(0));
```

✍ A l'aide d'IntelliJ, générez les accesseurs et mutateurs pour la propriété `currentBird`

✍ Ecrivez une méthode pour créer les liaisons de données entre composants et propriétés d'un oiseau (code similaire à celui écrit précédemment dans `initialize`) :

```
public void bind(Bird bird) {
    commonNameLabel.textProperty().bind(bird.commonNameProperty());
    latinNameLabel.textProperty().bind(bird.latinNameProperty());
    familyLabel.textProperty().bind(bird.familyProperty());
    genusLabel.textProperty().bind(bird.genusProperty());
    specieLabel.textProperty().bind(bird.specieProperty());
    descriptionLabel.textProperty().bind(bird.descriptionProperty());
    birdImageView.imageProperty().bind(bird.imageProperty());
}
```

✍ Modifiez la méthode `initialize` en conséquence :

```
public void initialize() {
    bind(getCurrentBird());
    birdListView.setCellFactory(new BirdCellFactory());
    birdListView.setItems(repository.birds);
}
```

✍ Et dans cette même méthode, ajoutez la liaison de données vers la sélection courante :

```
birdListView.setItems(repository.birds);
currentBirdProperty().bind(birdListView.getSelectionModel().
    selectedItemProperty());
```

✍ Lancez l'application. Les informations sur l'oiseau ne changent pas si l'utilisateur sélectionne un oiseau différent. Pourquoi ?

👉 Il faut recréer les liaisons de données lorsque l'oiseau courant change. Modifiez initialize pour écouter les modifications de l'oiseau courant et recréer les liaisons de données en conséquence :

```
public void initialize() {
    birdListView.setCellFactory(new BirdCellFactory());
    birdListView.setItems(repository.birds);
    currentBirdProperty().bind(birdListView.getSelectionModel().
        selectedItemProperty());
    currentBirdProperty().addListener((observable, oldBird, newBird) -> {
        // Pour une liaison unidirectionnelle, il n'est pas nécessaire de supprimer
        // l'ancienne liaison avant d'en créer une nouvelle
        if (newBird != null)
            bind(newBird);
    });
}
```

👉 Lancez l'application. Les informations sur l'oiseau changent si l'utilisateur sélectionne un oiseau différent. Mais initialement, la sélection dans la liste est nulle et les composants affichent des informations par défaut. Nous allons rendre invisible ces composants si la sélection est vide.

👉 Dans la classe **MainController**, ajoutez un variable annotée pour référence le composant VBox racine qui affiche les informations sur un oiseau :

```
@FXML
private VBox birdView;
```

👉 Dans **main-view.fxml**, associez birdView à **fx:id** pour la Vbox

👉 Créez une liaison de données entre la propriété **Visible** de la Vbox et la sélection courante :

```
public void initialize() {
    birdListView.setCellFactory(new BirdCellFactory());
    birdListView.setItems(repository.birds);
    currentBirdProperty().bind(birdListView.getSelectionModel().
        selectedItemProperty());
```

```
currentBirdProperty().addListener((observable, oldBird, newBird) -> {  
    // Pour une liaison unidirectionnelle, il n'est pas nécessaire de supprimer  
    // l'ancienne liaison avant d'en créer une nouvelle  
    if (newBird != null)  
        bind(newBird);  
});  
birdView.visibleProperty().bind(currentBirdProperty().isNotNull());  
// ou  
//birdView.visibleProperty().bind(Bindings.createBooleanBinding(  
//    () -> getCurrentBird() != null, currentBirdProperty()));  
}
```

👉 Lancez l'application. Aucune information n'est visible si la sélection est vide.

2. FENÊTRE DE DIALOGUE MODALE, COMMANDES

Nous allons donner la possibilité à l'utilisateur d'éditer un oiseau. L'édition se fera dans une fenêtre de dialogue. L'entrée en mode d'édition s'effectue en cliquant sur un bouton de la barre de commandes.

2.1. CRÉATION DU DIALOGUE D'ÉDITION

Pour simplifier, l'utilisateur ne pourra modifier que le nom commun et la description de l'oiseau.

 Dans la classe Bird, ajoutez une méthode pour recopier l'état d'un autre oiseau :

```
public void copyFrom(Bird bird) {
    setFamily(bird.getFamily());
    setGenus(bird.getGenus());
    setSpecie(bird.getSpecie());
    setCommonName(bird.getCommonName());
    setLatinName(bird.getLatinName());
    setDescription(bird.getDescription());
    setImagePath(bird.getImagePath());
}
```

 Dans **resources.ensisa.birds**, créez un fichier fxml nommé **editor-pane.fxml**

 Ecrivez le contenu de **editor-pane.fxml** (dans **Scene Builder**, c'est mal supporté)

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

<?import javafx.scene.control.ButtonType?>
<?import javafx.scene.control.DialogPane?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>
```

```

<DialogPane prefWidth="600.0" xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1">
    <content>
        <GridPane hgap="5.0" vgap="5.0">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" minWidth="-Infinity" />
                <ColumnConstraints hgrow="ALWAYS" minWidth="10.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0" valignment="BASELINE" />
                <RowConstraints minHeight="10.0" valignment="BASELINE" />
            </rowConstraints>
            <children>
                <Label text="Nom" />
                <Label text="Description" GridPane.rowIndex="1" />
                <TextField fx:id="nameTextField" GridPane.columnIndex="1" />
                <TextArea fx:id="descriptionTextArea" wrapText="true" prefHeight="200.0"
                           GridPane.columnIndex="1" GridPane.rowIndex="1" />
            </children>
        </GridPane>
    </content>
    <buttonTypes>
        <ButtonType fx:constant="CANCEL" />
        <ButtonType fx:constant="APPLY" />
    </buttonTypes>
</DialogPane>

```

 Créez une classe **BirdEditDialog**. Cette classe spécialise la classe **Dialog** de JavaFX paramétrée par le type de donnée à retourner. Nous retournons un oiseau modifié.

```

public class BirdEditDialog extends Dialog<Bird> {
}

```

 Déclarez des variables annotées pour référencer les composants du dialogue et une variable pour mémoriser l'oiseau en cours d'édition. Nous mémorisons une copie de l'oiseau car l'utilisateur peut annuler ses changements.

```

@FXML
private TextField nameTextField;
@FXML
private TextArea descriptionTextArea;

private Bird editedBird;

```

 Ecrivez le constructeur :

```

public BirdEditDialog(Window owner, Bird bird) {
    try {
        editedBird = new Bird();
        editedBird.copyFrom(bird);
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("/ensisa/birds/editor-pane.fxml"));
        loader.setController(this);

        DialogPane dialogPane = loader.load();
        initOwner(owner);
        initModality(Modality.APPLICATION_MODAL);

        setResizable(true);
        setTitle("Edition de " + editedBird.getCommonName());
        setDialogPane(dialogPane);
        setResultConverter(buttonType -> {
            if(!Objects.equals(ButtonBar.ButtonData.APPLY,
                buttonType.getButtonData())))
            {
                return null;
            }

            return editedBird;
        });

        setOnShowing(dialogEvent -> Platform.runLater(
            () -> nameTextField.requestFocus()));
    }
    catch (IOException e){
        throw new RuntimeException(e);
    }
}

```

```

        }
    }

public void initialize() {
    nameTextField.textProperty().bindBidirectional(
        editedBird.commonNameProperty());
    descriptionTextArea.textProperty().bindBidirectional(
        editedBird.descriptionProperty());
}

```

 Dans la classe **MainController**, déclarez une variable annotée pour référencer le bouton d'édition qui figurera dans la barre de commandes, et une méthode action correspondante :

```

@FXML
private Button editButton;

@FXML
private void editButtonAction(ActionEvent event) {
}

```

 Ecrivez le corps de la méthode ;

```

@FXML
private void editButtonAction(ActionEvent event) {
    Node node = (Node) event.getSource();
    Stage stage = (Stage) node.getScene().getWindow();
    BirdEditDialog dialog = new BirdEditDialog(stage, getCurrentBird());
    dialog.showAndWait().ifPresent(bird -> {
        getCurrentBird().copyFrom(bird);
    });
}

```

 Dans **main-view.fxml**, modifiez le titre du bouton qui figure dans la barre de commandes en "Editer", son **fx:id** en **editButton** et la méthode action en **editButtonAction**

 Lancez l'application. Sélectionnez un oiseau et appuyez sur le bouton d'édition. Sur l'appui sur le bouton **Appliquer** de la fenêtre de dialogue, les modifications sont répercutées sur l'oiseau sélectionné et l'affichage mis à jour grâce aux liaisons de données.

Remarque : une exception `NullPointerException` est levée si l'on appuie sur le bouton alors qu'aucun oiseau n'est sélectionné.

Le bouton d'édition doit être inactif si aucun oiseau n'est sélectionné.

 Dans la méthode `initialize` de `MainController`, ajoutez une liaison de données pour désactiver le bouton selon l'état de la sélection

```
editButton.disableProperty().bind(currentBirdProperty().isNull());
```

 Lancez l'application et vérifiez le changement d'état du bouton.

2.2. COMMANDE DE SUPPRESSION

 Dans la classe `MainController`, déclarez une variable annotée pour référencer le bouton de suppression qui figurera dans la barre de commandes, et une méthode action correspondante :

```
@FXML  
private Button deleteButton;  
  
@FXML  
private void deleteButtonAction(ActionEvent event) {  
  
}
```

 Ecrivez le corps de la méthode `deleteButtonAction` pour supprimer l'oiseau sélectionné du modèle

```
@FXML  
private void deleteButtonAction(ActionEvent event) {  
    repository.birds.remove(getCurrentBird());
```

```
}
```

✍ Créez une liaison de données pour désactiver le bouton de suppression si aucun oiseau n'est sélectionné :

```
deleteButton.disableProperty().bind(currentBirdProperty().isNull());
```

✍ Dans **main-view.fxml**, ajoutez un bouton à la barre de commandes, modifiez le titre du bouton en "Supprimer", son **fx:id** en **deleteButton** et la méthode action en **deleteButtonAction**

✍ Lancez l'application et vérifiez le fonctionnement de la suppression.

2.3. FILTRAGE DE LA LISTE

Nous allons donner la possibilité à l'utilisateur de filtrer la liste des oiseaux. Le texte du filtre sera recherché dans le nom commun des oiseaux. La classe **ObservableList** de JavaFX propose une méthode qui réalise un filtrage à l'aide d'un prédictat.

✍ Dans la classe **MainController**, déclarez une variable annotée pour référencer le champ de saisie qui figurera dans la barre de commandes :

```
@FXML  
private TextField filterTextField;
```

✍ Editez **main-view.fxml** dans **SceneBuilder**

Ajoutez un **TextField** dans la barre de commandes. Mettez "Filtre" dans **Prompt Text**, **filterTextField** dans **fx:id**.

✍ Dans la classe **MainController**, déclarez une variable d'instance pour mémoriser la liste filtrée des oiseaux :

```
public class MainController {  
    private final ObjectProperty<Bird> currentBird;  
    private BirdRepository repository;
```

```
private FilteredList<Bird> filteredBirdList;
```

✍ Dans la méthode `initialize`, créez la liste filtrée avec un prédictat par défaut et associez-la au composant `ListView` :

```
public void initialize() {
    birdListView.setCellFactory(new BirdCellFactory());
    filteredBirdList = new FilteredList<>(repository.birds);
    birdListView.setItems(filteredBirdList);
```

✍ Puis ajoutez un écouteur à la propriété `Text` du champ de saisie, pour modifier le prédictat de filtrage en conséquence :

```
public void initialize() {
    birdListView.setCellFactory(new BirdCellFactory());
    filteredBirdList = new FilteredList<>(repository.birds);
    birdListView.setItems(filteredBirdList);
    currentBirdProperty().bind(birdListView.getSelectionModel().
        selectedItemProperty());
    currentBirdProperty().addListener((observable, oldBird, newBird) -> {
        // Pour une liaison unidirectionnelle, il n'est pas nécessaire de supprimer
        // l'ancienne liaison avant d'en créer une nouvelle
        if (newBird != null)
            bind(newBird);
    });
    birdView.visibleProperty().bind(Bindings.createBooleanBinding(
        () -> getCurrentBird() != null, currentBirdProperty()));
    editButton.disableProperty().bind(Bindings.createBooleanBinding(
        () -> getCurrentBird() == null, currentBirdProperty()));
    deleteButton.disableProperty().bind(Bindings.createBooleanBinding(
        () -> getCurrentBird() == null, currentBirdProperty()));
    filterTextField.textProperty().addListener((observable, oldText, newText) -> {
        Predicate<Bird> filter = bird -> {
            String f = newText.trim().toLowerCase();
            return f.isEmpty() || bird.getCommonName().toLowerCase().contains(f);
        };
        filteredBirdList.setPredicate(filter);
    });
}
```

}

 Lancez l'application

Des oiseaux

Editor Supprimer |

Merle noir Turdus merula	Mésange bleue Cyanistes caeruleus
Pouillot véloce Phylloscopus collybita	
Mésange bleue Cyanistes caeruleus	<i>Paridae</i> <i>Cyanistes</i> <i>Caeruleus</i>
Rougegorge familier Erithacus rubecula	La Mésange bleue est une petite mésange qui tire son nom de la couleur bleue de sa calotte, de ses ailes et de sa queue. Sa tête est remarquable. La face, largement blanche, est barrée de trois traits bleu sombre à noirs, deux traits loraux qui passent par l'œil pour rejoindre la nuque de même couleur, et un large trait gulaire qui rejoint un collier, qui lui-même borde les joues blanches et rejoint la nuque. Le mâle adulte se distingue à la teinte bleue du plumage plus marquée, surtout à la calotte. Le bleu de la femelle est plus terne. Le manteau est jaune-vert. Un trait blanc se voit sur l'aile au niveau des grandes couvertures. Les parties inférieures sont jaune-citron, avec une esquisse de ligne médiobrachiale noirâtre sur le bas de la poitrine et le haut du ventre, rappelant un peu celle de sa cousine charbonnière, mais moins marquée. L'œil est sombre. Bec et pattes sont gris bleuté.
Pouillot fitis Phylloscopus trochilus	
Troglodyte mignon Troglodytes troglodytes	
Rossignol philomèle Luscinia megarhynchos	
Alouette des champs Alauda arvensis	
Chardonneret élégant Carduelis carduelis	
Roitelet huppé Regulus regulus	
Rousserolle verderolle Acrocephalus palustris	

< >

1. conteneurs, liste	1
1.1.Création du projet	2
1.2.Le modèle	3
1.3.Ebauche de la vue de détail sur un oiseau	5
1.4.Modèle actif	8
1.5.La vue de détail de l'oiseau, suite...	9
1.6.style	12
1.7.Lecture d'une image en ressources	12
1.8.Ajout de l'affichage d'une liste d'oiseaux	15
1.9.Prise en compte de la sélection d'un oiseau	18
2. Fenêtre de dialogue modale, Commandes	22
2.1.Création du dialogue d'édition	22
2.2.Commande de suppression	26
2.3.Filtrage de la liste	27

