

## JavaFX

Utilisation des contrôles d'interface utilisateur JavaFX

Version 2.2

**E20485-11**

septembre 2013

Dans ce didacticiel, vous apprendrez à créer des interfaces utilisateur dans vos applications JavaFX avec les contrôles d'interface utilisateur disponibles via l'API JavaFX.

JavaFX/Utilisation des contrôles d'interface utilisateur JavaFX, version 2.2

E20485-11

Copyright © 2011, 2013 Oracle et/ou ses filiales. Tous les droits sont réservés.

Auteur principal: Alla Redko

Ce logiciel et la documentation associée sont fournis dans le cadre d'un contrat de licence contenant des restrictions d'utilisation et de divulgation et sont protégés par les lois sur la propriété intellectuelle. Sauf dans les cas expressément autorisés dans votre contrat de licence ou autorisés par la loi, vous ne pouvez pas utiliser, copier, reproduire, traduire, diffuser, modifier, concéder sous licence, transmettre, distribuer, exposer, exécuter, publier ou afficher toute partie, sous quelque forme que ce soit, ou n'importe comment. L'ingénierie inverse, le désassemblage ou la décompilation de ce logiciel, à moins que la loi ne l'exige pour l'interopérabilité, est interdit.

Les informations contenues dans ce document sont sujettes à modification sans préavis et ne sont pas garanties sans erreur. Si vous trouvez des erreurs, veuillez nous les signaler par écrit.

Si ce logiciel ou la documentation connexe est livré au gouvernement des États-Unis ou à toute personne l'autorisant au nom du gouvernement des États-Unis, l'avis suivant s'applique:

**DROITS DU GOUVERNEMENT DES ÉTATS-UNIS** Les programmes, logiciels, bases de données, ainsi que la documentation et les données techniques connexes fournies aux clients du gouvernement des États-Unis sont des «logiciels informatiques commerciaux» ou des «données techniques commerciales» conformément à la réglementation fédérale sur les acquisitions applicable et aux réglementations supplémentaires spécifiques aux agences. À ce titre, l'utilisation, la duplication, la divulgation, la modification et l'adaptation sont soumises aux restrictions et aux conditions de licence énoncées dans le contrat gouvernemental applicable et, dans la mesure où les conditions du contrat gouvernemental s'appliquent, aux droits supplémentaires énoncés dans FAR 52.227-19, Licence de logiciel informatique commercial (décembre 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

Ce logiciel est développé pour une utilisation générale dans une variété d'applications de gestion de l'information. Il n'est pas développé ou destiné à être utilisé dans des applications intrinsèquement dangereuses, y compris des applications pouvant créer un risque de blessure. Si vous utilisez ce logiciel dans des applications dangereuses, vous serez responsable de prendre toutes les mesures appropriées de sécurité, de sauvegarde, de redondance et autres pour garantir l'utilisation en toute sécurité de ce logiciel. Oracle Corporation et ses filiales déclinent toute responsabilité pour tout dommage causé par l'utilisation de ce logiciel dans des applications dangereuses.

Oracle est une marque déposée d'Oracle Corporation et/ou de ses filiales. Les autres noms peuvent être des marques déposées de leurs propriétaires respectifs.

Ce logiciel et cette documentation peuvent fournir un accès ou des informations sur le contenu, les produits et les services de tiers. Oracle Corporation et ses sociétés affiliées ne sont pas responsables et déclinent expressément toute garantie de quelque nature que ce soit concernant le contenu, les produits et les services de tiers. Oracle Corporation et ses sociétés affiliées ne seront pas responsables des pertes, coûts ou dommages encourus en raison de votre accès ou de votre utilisation de contenus, produits ou services tiers.

# Contenu

## Partie I À propos de ce didacticiel

### 1 Contrôles de l'interface utilisateur JavaFX

Contrôles d'interface utilisateur pris en charge dans JavaFX 2 .....	1-1
Fonctionnalités et effets .....	1-2
Styliser les contrôles de l'interface utilisateur avec CSS.....	1-3
Graphiques.....	1-4
Intégration des contrôles d'interface utilisateur JavaFX 2 dans Swing.....	1-4

### 2 Étiquette

Création d'une étiquette .....	2-1
Définition d'une police .....	2-2
Habillement du texte .....	2-2
Application d' effets .....	2-3

### 3 Bouton

Création d'un bouton .....	3-1
Attribution d'une action.....	3-2
Application d' effets .....	3-2
Styliser un bouton .....	3-3

### 4 bouton radio

Création d'un bouton radio .....	4-1
Ajouter des boutons radio aux groupes .....	4-2

Traitement des événements pour les boutons radio.....	4-2
Demander le focus pour un bouton radio.....	4-3
<b>5 Bouton bascule</b>	
Création d'un bouton bascule .....	5-1
Ajouter des boutons bascule à un groupe .....	5-2
Paramétriser le comportement .....	5-2
Boutons à bascule de style .....	5-3
<b>6 Case à cocher</b>	
Création de cases à cocher .....	6-1
Définition d'un état .....	6-1
Paramétriser le comportement .....	6-2
Styliser une case à cocher .....	6-3
<b>Boîte à 7 choix</b>	
Création d'une boîte à choix .....	7-1
Paramétriser le comportement d'une boîte de choix.....	7-2
Application d'une info-bulle .....	7-3
<b>8 Champ de texte</b>	
Création d'un champ de texte .....	8-1
Construire l'interface utilisateur avec des champs de texte .....	8-1
Traitement des données de champ de texte.....	8-3
<b>9 Barre de défilement</b>	
Création d'une barre de défilement .....	9-1
Utilisation d'une barre de défilement dans votre application .....	9-2
<b>10 Volet de défilement</b>	
Création d'un volet de défilement .....	10-1
Définition de la stratégie de barre de défilement pour un volet de défilement.....	10-1
Redimensionnement des composants dans le volet de défilement .....	10-2
Exemple d'application avec un volet de défilement .....	dix- 2
<b>11 Vue Liste</b>	
Création d'une vue de liste.....	11-1
Remplir une vue de liste avec des données .....	11-2
Personnalisation du contenu d'une vue de liste.....	11- 4
Traitement de la sélection d'éléments de liste .....	11-6
<b>12 Affichage du tableau</b>	
Création d'un tableau .....	12-2
Définition du modèle de données .....	12-4

<b>Ajout de nouvelles lignes .....</b>	<b>12-8</b>
<b>Trier les données en colonnes .....</b>	<b>12-13</b>
<b>Modification des données dans le tableau .....</b>	<b>12-13</b>
<b>Ajout de cartes de données à la table.....</b>	<b>12-23</b>
<b>13 Arborescence</b>	
<b>Création d'arborescences .....</b>	<b>13-1</b>
<b>Mise en place des Usines Cellulaires .....</b>	<b>13-3</b>
<b>Ajout de nouveaux éléments d'arborescence à la demande.....</b>	<b>13-9</b>
<b>Utilisation des éditeurs de cellules d'arborescence.....</b>	<b>13-14</b>
<b>14 Boîte combinée</b>	
<b>Création de listes déroulantes .....</b>	<b>14-1</b>
<b>Listes déroulantes modifiables .....</b>	<b>14-4</b>
<b>Application des fabriques de cellules aux zones de liste déroulante .....</b>	<b>14-7</b>
<b>15 Séparateur</b>	
<b>Création d'un séparateur .....</b>	<b>15-1</b>
<b>Ajout de séparateurs à l'interface utilisateur de votre application .....</b>	<b>15-2</b>
<b>Séparateurs de style .....</b>	<b>15-4</b>
<b>16 Curseur</b>	
<b>Création d'un curseur .....</b>	<b>16-1</b>
<b>Utilisation des curseurs dans les applications graphiques.....</b>	<b>16-2</b>
<b>17 Barre de progression et indicateur de progression</b>	
<b>Création de contrôles de progression .....</b>	<b>17-1</b>
<b>Affichage de la progression dans votre interface utilisateur .....</b>	<b>17-3</b>
<b>18 Hyperlien</b>	
<b>Création d'un lien hypertexte.....</b>	<b>18-1</b>
<b>Liaison du contenu local .....</b>	<b>18-1</b>
<b>Liaison du contenu distant .....</b>	<b>18-4</b>
<b>19 Éditeur HTML</b>	
<b>Ajout d'un éditeur HTML .....</b>	<b>19-2</b>
<b>Utilisation d'un éditeur HTML pour créer l' interface utilisateur.....</b>	<b>19-4</b>
<b>Obtention du contenu HTML .....</b>	<b>19-6</b>
<b>20 Info-bulle</b>	
<b>Création d'une info- bulle.....</b>	<b>20-1</b>
<b>Présentation des données d'application dans les info-bulles .....</b>	<b>20-2</b>

**21 Volet titré et accordéon**

<b>Création de volets titrés .....</b>	<b>21-1</b>
<b>Ajouter des volets titrés à un accordéon .....</b>	<b>21-3</b>
<b>Traitement des événements pour un accordéon avec des volets titrés .....</b>	<b>21-4</b>

**22 menus**

<b>Création de menus dans les applications JavaFX .....</b>	<b>22-2</b>
<b>Création d'une barre de menus .....</b>	<b>22-2</b>
<b>Ajout d' éléments de menu.....</b>	<b>22-5</b>
<b>Création de sous-menus .....</b>	<b>22-10</b>
<b>Ajouter des menus contextuels .....</b>	<b>22-13</b>

**23 Champ Mot de passe**

<b>Création d'un champ de mot de passe .....</b>	<b>23-1</b>
<b>Évaluation du mot de passe.....</b>	<b>23-2</b>

**Sélecteur de 24 couleurs**

<b>Aperçu de la conception .....</b>	<b>24-1</b>
<b>Sélecteur de couleurs .....</b>	<b>24-2</b>
<b>Palette de couleurs.....</b>	<b>24-2</b>
<b>Fenêtre de dialogue de couleur personnalisée .....</b>	<b>24-3</b>
<b>Utilisation d'un sélecteur de couleurs .....</b>	<b>24-4</b>
<b>Modification de l'apparence d'un sélecteur de couleurs.....</b>	<b>24-8</b>

**25 Contrôle de la pagination**

<b>Création d'un contrôle de pagination .....</b>	<b>25-1</b>
<b>Implémentation des fabriques de pages .....</b>	<b>25-3</b>
<b>Styliser un contrôle de pagination.....</b>	<b>25-9</b>

**26 Sélecteur de fichiers**

<b>Ouverture de fichiers .....</b>	<b>26-1</b>
<b>Configuration d'un sélecteur de fichiers .....</b>	<b>26-6</b>
<b>Paramétrage des filtres d'extension .....</b>	<b>26-9</b>
<b>Enregistrement de fichiers .....</b>	<b>26-11</b>

**27 Personnalisation des commandes de l'interface utilisateur**

<b>Application du CSS.....</b>	<b>27-1</b>
<b>Modification du comportement par défaut .....</b>	<b>27-4</b>
<b>Mise en place des Usines Cellulaires .....</b>	<b>27-6</b>

## Première partie

---

---

### À propos de ce didacticiel

Ce didacticiel couvre les contrôles d'interface utilisateur JavaFX intégrés disponibles dans l'API JavaFX.

Le document contient les chapitres suivants:

- ÿ Contrôles d'interface utilisateur JavaFX
- ÿ Étiquette
- ÿ Bouton
- ÿ Bouton radio
- ÿ Bouton bascule
- ÿ Case à cocher
- ÿ Boîte de choix
- ÿ Champ de texte
- ÿ Champ de mot de passe
- ÿ Barre de défilement
- ÿ Volet de défilement
- ÿ Affichage en liste
- ÿ Affichage du tableau
- ÿ Arborescence
- ÿ Boîte combinée
- ÿ Séparateur
- ÿ Curseur
- ÿ Barre de progression et indicateur de progression
- ÿ Lien hypertexte ÿ Info -bulle
- ÿ Éditeur HTML
- ÿ Volet titré et accordéon
- ÿ Menus
- ÿ Sélecteur de couleurs
- ÿ Contrôle de la pagination
- ÿ Sélecteur de fichiers

ÿ [Personnalisation des contrôles de l'interface utilisateur](#)

Chaque chapitre fournit des exemples de code et des applications pour illustrer le fonctionnement d'un contrôle d'interface utilisateur particulier. Vous pouvez trouver les fichiers source des applications et les projets NetBeans correspondants dans la table des matières.

## 1Contrôles de l'interface utilisateur JavaFX

Ce chapitre fournit un aperçu des contrôles de l'interface utilisateur JavaFX disponibles via l'API.

Les contrôles de l'interface utilisateur JavaFX sont créés à l'aide de nœuds dans le graphe de scène. Par conséquent, les contrôles peuvent utiliser les fonctionnalités visuellement riches de la plate-forme JavaFX. Étant donné que les API JavaFX sont entièrement implémentées en Java, vous pouvez facilement intégrer les contrôles de l'interface utilisateur JavaFX dans vos applications Java existantes.

La [Figure 1–1](#) montre les contrôles d'interface utilisateur typiques que vous pouvez trouver dans l'exemple d'application Ensemble. Essayez cette application pour évaluer le large éventail de contrôles, leur comportement et les styles disponibles.

Figure 1–1 Contrôles de l'interface utilisateur JavaFX



### Contrôles d'interface utilisateur pris en charge dans JavaFX 2

Les classes pour construire les contrôles de l'interface utilisateur résident dans le package `javafx.scene.control` de l'API.

La liste des contrôles d'interface utilisateur comprend des composants d'interface utilisateur typiques que vous pouvez reconnaître dans votre développement précédent d'applications clientes en Java. Cependant, le SDK JavaFX 2

introduit de nouveaux contrôles d'interface utilisateur Java, tels que TitledPane, ColorPicker et Pagination.

La [figure 1–2](#) montre une capture d'écran de trois éléments TitledPane avec des listes de paramètres pour une application de réseau social. Les listes peuvent glisser vers l'intérieur (rétracter) et glisser vers l'extérieur (étendre).

**Figure 1–2 Volets titrés**



Consultez la documentation de l'API pour la liste complète des contrôles de l'interface utilisateur.

Les classes de contrôle de l'interface utilisateur fournissent des variables et des méthodes supplémentaires au-delà de celles de la classe Control pour prendre en charge les interactions utilisateur typiques de manière intuitive. Vous pouvez attribuer un style spécifique à vos composants d'interface utilisateur en appliquant des feuilles de style en cascade (CSS). Pour certaines tâches inhabituelles, vous devrez peut-être étendre la classe Control pour créer un composant d'interface utilisateur personnalisé ou utiliser l'interface Skin pour définir un nouvel habillage pour un contrôle existant.

## Caractéristiques et effets

Étant donné que les contrôles d'interface utilisateur du package `javafx.scene.control` sont tous des extensions de la classe Node, ils peuvent être intégrés au rendu, à l'animation, aux transformations et aux transitions animées du graphe de scène.

Considérez la tâche consistant à créer un bouton, à lui appliquer une réflexion et à animer le bouton en modifiant son opacité de sa valeur maximale à sa valeur minimale.

La [Figure 1–3](#) montre trois états du bouton dans la chronologie de l'animation. L'image de gauche montre le bouton lorsque son opacité est réglée sur 1,0, l'image centrale montre l'opacité réglée sur 0,8 et l'image de droite montre l'opacité réglée sur 0,5.

**Figure 1–3 Bouton animé**



En utilisant les API JavaFX, vous pouvez implémenter cette tâche avec seulement quelques lignes de code.

L' [exemple 1–1](#) crée et démarre une chronologie indéfinie, dans laquelle, dans une image clé de 600 millisecondes, l'opacité du bouton passe de sa valeur par défaut (1,0) à 0,0. La méthode `setAutoReverse` active l'ordre inverse.

### Exemple 1–1 Cr éation d'un bouton animé

```
importer javafx.animation.KeyFrame;
importer javafx.animation.KeyValue;
importer javafx.animation.Timeline;
import javafx.util.Duration;
```

```

importer javafx.scene.control.Button; importer
javafx.scene.text.Font; importer
javafx.scene.effect.Reflection;

...
Bouton bouton = nouveau Bouton();
bouton.setText("OK");
button.setFont(nouvelle police("Tahoma", 24));
button.setEffect(nouvelle réflexion());

Chronologie finale timeline = new Timeline();
chronologie.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true); KeyValue finale kv = new
KeyValue(button.opacityProperty(), 0); KeyFrame final kf = new KeyFrame(Duration.millis(600),
kv); timeline.getKeyFrames().add(kf); chronologie.play();

...

```

Vous pouvez également appliquer d'autres effets visuels disponibles dans le package `javafx.scene.effect`, tels que l'ombre, l'éclairage ou le flou de mouvement.

### Styliser les contrôles de l'interface utilisateur avec CSS

Vous pouvez personnaliser l'apparence des commandes d'interface utilisateur intégrées en définissant vos propres feuilles de style en cascade (CSS). L'utilisation de CSS dans les applications JavaFX est sensiblement la même que l'utilisation de CSS dans HTML, car chaque cas est basé sur la même spécification CSS. L'état visuel d'un contrôle est défini par le fichier `.css`, comme illustré dans l' [exemple 1–2](#).

#### Exemple 1–2 Définition de styles pour les contrôles de l'interface utilisateur dans le fichier CSS

```

/*controlStyle.css */

.scène{
    -fx-police : 14pt "Cambria Bold"; -fx-color :
    #e79423 ;
    -fx-fond: #67644e;
}

.bouton{
    -fx-text-fill: #006464; -fx-
    background-color: #e79423; -fx-bordure-
    rayon: 20px;
    -rayon d'arrière-plan fx: 20px; -fx-
    rembourrage: 5px;
}

```

Vous pouvez activer le style dans l'application via la méthode `getStylesheets` de la classe `Scene`, comme illustré dans l' [exemple 1–3](#).

#### Exemple 1–3 Application de CSS

```

Scène scène = nouvelle Scène();
scène.getStylesheets().add("uicontrolssample/controlStyle.css");

```

De plus, vous définissez le style d'un contrôle directement dans le code de votre application en utilisant la méthode `setStyle`. La propriété `-fx-base` définie pour le bouton bascule dans l' [exemple 1–4](#) remplace les propriétés correspondantes définies dans le fichier `.css` pour tous les contrôles ajoutés à la scène.

**Exemple 1–4 Définition du style d'un bouton bascule dans l'application JavaFX**

```
ToggleButton tb3 = nouveau ToggleButton ("Je ne sais pas");
tb3.setStyle("-fx-base: #ed1c24;");
```

La [Figure 1–4](#) montre à quoi ressemble le bouton bascule stylisé lorsqu'il est ajouté à l'application.

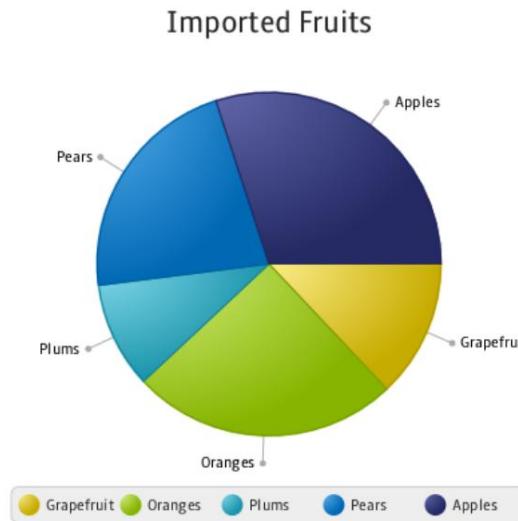
**Figure 1–4 Application du style CSS à un bouton bascule**

**Graphiques**

En plus des éléments typiques d'une interface utilisateur, JavaFX SDK fournit des graphiques préfabriqués dans le package `javafx.scene.chart`. Les types de graphiques suivants sont actuellement pris en charge: graphique en aires, graphique à barres, graphique à bulles, graphique linéaire, graphique à secteurs et graphique en nuage de points. Un graphique peut contenir plusieurs séries de données.

La [figure 1–5](#) montre un diagramme circulaire des fruits importés.

**Figure 1–5 Diagramme à secteurs**



Contrairement aux autres kits d'outils client Java, avec le SDK JavaFX, vous pouvez créer un tel graphique dans votre application en ajoutant seulement quelques lignes de code. Vous pouvez également définir divers schémas de couleurs et styles, appliquer des effets visuels, traiter des événements de souris et créer une animation.

Voir [Utilisation des graphiques JavaFX](#) pour plus d'informations sur les fonctionnalités et les capacités des graphiques.

**Intégration des contrôles d'interface utilisateur JavaFX 2 dans Swing**

Vous pouvez intégrer les contrôles de l'interface utilisateur JavaFX dans les applications clientes Java existantes basées sur la boîte à outils Swing.

**Pour intégrer du contenu JavaFX dans une application Swing, procédez comme suit:**

**1.** Ajoutez tous les contrôles JavaFX UI à l'objet `javafx.scene.Scene` un par un, dans un conteneur de mise en page ou en tant que groupe.

**2.** Ajoutez l'objet `Scene` au contenu de l'application Swing.

Si vous devez placer un seul contrôle JavaFX 2 dans votre code Swing existant, vous devez effectuer les deux étapes précédentes.

Même lorsqu'ils sont intégrés dans une application Swing, les contrôles de l'interface utilisateur JavaFX 2 sont toujours rendus à l'aide de la bibliothèque graphique Prism et tirent pleinement parti de ses capacités de rendu avancées.

Consultez le didacticiel JavaFX dans Swing pour plus d'informations sur l'interopérabilité de JavaFX et Swing.



# 2

## 2 Étiquette

Ce chapitre explique comment utiliser la classe Label qui réside dans le package javafx.scene.control de l'API JavaFX pour afficher un élément de texte. Apprenez à envelopper un élément de texte pour l'adapter à l'espace spécifique, ajouter une image graphique ou appliquer des effets visuels.

[La Figure 2–1](#) illustre trois utilisations courantes des étiquettes. L'étiquette à gauche est un élément de texte avec une image, l'étiquette au centre représente le texte pivoté et l'étiquette à droite affiche le texte enveloppé.

**Figure 2–1 Exemple d'application avec étiquettes**



### Création d'une étiquette

L'API JavaFX fournit trois constructeurs de la classe Label pour créer des étiquettes dans votre application, comme illustré dans l' [exemple 2–1](#).

#### Exemple 2–1 Crédit d'étiquettes

```
//Une étiquette vide
Étiquette étiquette1 = nouvelle étiquette();
// Une étiquette avec l'élément de texte
Label label2 = new Label("Rechercher");
// Une étiquette avec l'élément de texte et l'icône graphique
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
Label label3 = new Label("Recherche", new ImageView(image));
```

Une fois que vous avez créé une étiquette dans votre code, vous pouvez y ajouter du contenu textuel et graphique en utilisant les méthodes suivantes de la classe Labeled.

- ÿ La méthode `setText(String text)` – spécifie la légende de texte pour l'étiquette
- ÿ `setGraphic(Node graphic)` – spécifie l'icône graphique

La méthode `setTextFill` spécifie la couleur pour peindre l'élément de texte de l'étiquette.

Exemple d'étude [2-2](#). Il crée une étiquette de texte, y ajoute une icône et spécifie une couleur de remplissage pour le texte.

#### **Exemple 2-2 Ajout d'une icône et d'un remplissage de texte à une étiquette**

```
Étiquette étiquette1 = nouvelle étiquette("Rechercher");
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
label1.setGraphic(nouvelle ImageView(image));
label1.setTextFill(Color.web("#0076a3"));
```

Lorsque ce fragment de code est ajouté à l'application, il produit l'étiquette illustrée à la [Figure 2-2](#).

**Figure 2-2 Étiquette avec icône**



Lors de la définition du texte et du contenu graphique de votre bouton, vous pouvez utiliser la méthode `setGraphicTextGap` pour définir l'écart entre eux.

De plus, vous pouvez modifier la position du contenu de l'étiquette dans sa zone de mise en page à l'aide de la méthode `setTextAlignment`. Vous pouvez également définir la position du graphique par rapport au texte en appliquant la méthode `setContentDisplay` et en spécifiant l'une des constantes `ContentDisplay` suivantes : `LFFT`, `RIGHT`, `CENTER`, `TOP`, `BOTTOM`.

## Définition d'une police

Comparez les étiquettes de recherche dans la [Figure 2-1](#) et la [Figure 2-2](#). Notez que l'étiquette de la [Figure 2-1](#) a une taille de police plus grande. Cela est dû au fait que le fragment de code illustré dans l'[exemple 2-2](#) ne spécifie aucun paramètre de police pour l'étiquette. Il est rendu avec la taille de police par défaut.

Pour fournir une taille de texte de police autre que celle par défaut pour votre étiquette, utilisez la commande `setFont` méthode de la classe `Labeled`. Le fragment de code de l'[exemple 2-3](#) définit la taille du texte `label1` sur 30 points et le nom de la police sur Arial. Pour `label2`, définissez la taille du texte sur 32 points et le nom de la police sur Cambria.

#### **Exemple 2-3 Application des paramètres de police**

```
//Utilise un constructeur de la classe Font
label1.setFont(nouvelle police("Arial", 30));
//Utilise la méthode font de la classe Font
label2.setFont(Font.font("Cambria", 32));
```

## Habillage du texte

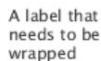
Lorsque vous créez une étiquette, vous devez parfois la faire tenir dans un espace plus petit que celui dont vous avez besoin pour le rendu. Pour décomposer (envelopper) le texte afin qu'il puisse tenir dans la zone de mise en page, définissez la valeur true pour la méthode `setWrapText`, comme illustré dans l'[exemple 2-4](#).

**Exemple 2–4 Activer l'habillage du texte**

```
Label label3 = new Label("Une étiquette qui doit être enveloppée");
label3.setWrapText(true);
```

Lorsque label3 est ajouté au contenu d'une application, il est rendu comme illustré à la [Figure 2–3](#).

**Figure 2–3 Étiquette avec texte enveloppé**



A label that  
needs to be  
wrapped

Supposons que la zone de mise en page de l'étiquette soit limitée non seulement par sa largeur, mais également par sa hauteur. Vous pouvez spécifier le comportement d'une étiquette lorsqu'il est impossible de restituer l'intégralité de la chaîne de texte requise. Utilisez la méthode setTextOverrun de la classe Labeled et l'un des types OverrunStyle disponibles pour définir comment traiter la partie de la chaîne de texte qui ne peut pas être restituée correctement. Consultez la documentation de l'API pour plus d'informations sur les types OverrunStyle.

## Appliquer des effets

Bien qu'une étiquette soit un contenu statique et ne puisse pas être modifiée, vous pouvez lui appliquer des effets visuels ou des transformations. Le fragment de code de l' [exemple 2–5](#) fait pivoter label2 de 270 degrés et translate sa position verticalement.

**Exemple 2–5 Rotation d'une étiquette**

```
Label label2 = nouveau Label ("Valeurs");
label2.setFont(nouvelle police("Cambria", 32));
label2.setRotate(270);
label2.setTranslateY (50);
```

La rotation et la traduction sont des transformations typiques disponibles dans l'API JavaFX.

De plus, vous pouvez configurer un effet qui zoomé (agrandit) l'étiquette lorsqu'un utilisateur passe le curseur de la souris dessus.

Le fragment de code illustré dans l' [exemple 2–6](#) applique l'effet de zoom à label3. Lorsque l'événement MOUSE\_ENTERED est déclenché sur l'étiquette, le facteur d'échelle de 1,5 est défini pour les méthodes setScaleX et setScaleY. Lorsqu'un utilisateur déplace le curseur de la souris hors de l'étiquette et que l'événement MOUSE\_EXITED se produit, le facteur d'échelle est défini sur 1,0 et l'étiquette est restituée dans sa taille d'origine.

**Exemple 2–6 Application de l'effet de zoom**

```
label3.setOnMouseEntered(new EventHandler<MouseEvent>() {
    @Override public void handle(MouseEvent e) {
        label3.setScaleX(1.5);
        label3.setScaleY(1.5);
    }
});

label3.setOnMouseExited(new EventHandler<MouseEvent>() {
    @Override public void handle(MouseEvent e) {
        label3.setScaleX(1);
        label3.setScaleY(1);
    }
});
```

[Appliquer des effets](#)

```
});
```

La [figure 2–4](#) montre les deux états de label3.

**Figure 2–4 Zoom sur une étiquette**



### Documentation relative à l'API

- ÿ Étiquette
- ÿ Étiqueté

# 3

## 3Bouton

La classe Button disponible via l'API JavaFX permet aux développeurs de traiter une action lorsqu'un utilisateur clique sur un bouton. La classe Button est une extension de la classe Labeled classer. Il peut afficher du texte, une image ou les deux. [La Figure 3-1](#) montre des boutons avec divers effets. Dans ce chapitre, vous apprendrez à créer chacun de ces types de boutons.

**Figure 3-1 Types de boutons**



### Création d'un bouton

Vous pouvez créer un contrôle Button dans une application JavaFX en utilisant trois constructeurs de la classe Button, comme illustré dans l' [exemple 3-1](#).

#### Exemple 3-1 Cr éation d'un bouton

```
// Un bouton avec une légende de texte vide.  
Bouton bouton1 = nouveau Bouton();  
// Un bouton avec la légende de texte spécifiée.  
Button button2 = new Button("Accepter");  
// Un bouton avec la légende de texte et l'icône spécifiées.  
Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));  
Button button3 = new Button("Accepter", new ImageView(imageOk));
```

Étant donné que la classe Button étend la classe Labeled, vous pouvez utiliser les méthodes suivantes pour spécifier le contenu d'un bouton qui n'a pas d'icône ou de légende:

- ÿ La méthode `setText(String text)` – spécifie la légende du texte pour le bouton
- ÿ La méthode `setGraphic(Node graphic)`– spécifie l'icône graphique

L'[exemple 3-2](#) montre comment créer un bouton avec une icône mais sans légende de texte.

#### Exemple 3-2 Ajout d'une icône à un bouton

```
Image imageDecline = new Image(getClass().getResourceAsStream("not.png"));  
Bouton bouton5 = nouveau Bouton();
```

## Attribuer une action

```
button5.setGraphic(new ImageView(imageDecline));
```

Lorsqu'il est ajouté à l'application, ce fragment de code produit le bouton illustré à la [Figure 3–2](#).

**Figure 3–2 Bouton avec icône**



Dans l' [exemple 3–2](#) et la [figure 3–2](#), l'icône est un objet ImageView. Cependant, vous pouvez utiliser d'autres objets graphiques, par exemple des formes qui résident dans le package javafx.scene.shape. Lors de la définition du texte et du contenu graphique de votre bouton, vous pouvez utiliser la méthode setGraphicTextGap pour définir l'écart entre eux.

L'habillage par défaut de la classe Button distingue les états visuels suivants du bouton. La [Figure 3–3](#) montre les états par défaut d'un bouton avec une icône.

**Figure 3–3 États des boutons**



## Attribuer une action

La fonction principale de chaque bouton est de produire une action lorsqu'il est cliqué. Utilisez la méthode setOnAction de la classe Button pour définir ce qui se passe lorsqu'un utilisateur clique sur le bouton. L'[exemple 3–3](#) montre un fragment de code qui définit une action pour button2.

### Exemple 3–3 Définition d'une action pour un bouton

```
button2.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
        label.setText("Accepté");
    }
});
```

ActionEvent est un type d'événement qui est traité par EventHandler. Un objet EventHandler fournit la méthode handle pour traiter une action déclenchée pour un bouton. L'[exemple 3–3](#) montre comment remplacer la méthode handle, de sorte que lorsqu'un utilisateur appuie sur le bouton 2, la légende de texte d'une étiquette est définie sur "Accepté".

Vous pouvez utiliser la classe Button pour définir autant de méthodes de gestion d'événements que nécessaire pour provoquer le comportement spécifique ou appliquer des effets visuels.

## Appliquer des effets

Étant donné que la classe Button étend la classe Node, vous pouvez appliquer n'importe quel effet du package javafx.scene.effect pour améliorer l'apparence visuelle du bouton.

Dans l'[exemple 3–4](#), l'effet DropShadow est appliqué au bouton3 lorsque l'événement onMouseEntered se produit.

### Exemple 3–4 Application de l'effet DropShadow

```
DropShadow shadow = new DropShadow();
//Ajout de l'ombre lorsque le curseur de la souris est activé
button3.addEventHandler(MouseEvent.MOUSE_ENTERED, new
    EventHandler<MouseEvent>() {
        @Override public void handle(MouseEvent e) {
            button3.setEffect(shadow);
        }
    });
//Suppression de l'ombre lorsque le curseur de la souris est désactivé
button3.addEventHandler(MouseEvent.MOUSE_EXITED, new
    EventHandler<MouseEvent>() {
        @Override public void handle(MouseEvent e) {
            button3.setEffect(null);
        }
    });
});
```

La [Figure 3–4](#) montre les états du bouton3 lorsque le curseur de la souris est dessus et lorsqu'il est éteint.

**Figure 3–4 Bouton avec ombre portée**



## Styliser un bouton

L'étape suivante pour améliorer l'apparence visuelle d'un bouton consiste à appliquer les styles CSS définis par la classe Skin. L'utilisation de CSS dans les applications JavaFX 2 est similaire à l'utilisation de CSS dans HTML, car chaque cas est basé sur la même spécification CSS.

Vous pouvez définir des styles dans un fichier CSS séparé et les activer dans l'application à l'aide de la méthode `setStyleClass`. Cette méthode est héritée de la classe `Node` et est disponible pour tous les contrôles de l'interface utilisateur. Vous pouvez également définir le style d'un bouton directement dans le code en utilisant la méthode `setStyle`. L'[exemple 3–5](#) et la [figure 3–5](#) illustrent cette dernière approche.

### Exemple 3–5 Styliser un bouton

```
button1.setStyle("-fx-font-size: 22 arial; -fx-base: #b6e7c9;");
```

La propriété `-fx-font-size` définit une taille de police pour le bouton1. La propriété `-fx-base` remplace la couleur par défaut appliquée au bouton. Par conséquent, le bouton 1 est vert clair avec une taille de texte plus grande, comme illustré à la [Figure 3–5](#).

**Figure 3–5 Bouton stylisé avec CSS**



## Documentation relative à l'API

ÿ Bouton

ÿ Étiqueté

Styliser un bouton

---

## 4

## 4Bouton radio

Ce chapitre traite du contrôle de bouton radio et de la classe `RadioButton`, une implémentation spécialisée de la classe `ToggleButton`.

Un contrôle de bouton radio peut être sélectionné ou désélectionné. Généralement, les boutons radio sont combinés dans un groupe où un seul bouton à la fois peut être sélectionné. Ce comportement les distingue des boutons bascule, car tous les boutons bascule d'un groupe peuvent être dans un état désélectionné.

La [Figure 4–1](#) montre trois captures d'écran de l'exemple `RadioButton`, dans lequel trois boutons radio sont ajoutés à un groupe.

**Figure 4–1 Exemple de bouton radio**



Étudiez les paragraphes suivants pour en savoir plus sur la façon d'implémenter les boutons radio dans vos applications.

### Création d'un bouton radio

La classe `RadioButton` disponible dans le package `javafx.scene.control` du SDK JavaFX fournit deux constructeurs avec lesquels vous pouvez créer un bouton radio.

L'[exemple 4–1](#) montre deux boutons radio. Le constructeur sans paramètre est utilisé pour créer `rb1`. La légende de texte de ce bouton radio est définie à l'aide de la méthode `setText`. La légende de texte pour `rb2` est définie dans le constructeur correspondant.

#### Exemple 4–1 Création de boutons radio

```
// Un bouton radio avec une chaîne vide pour son étiquette
RadioButton rb1 = new RadioButton();
//Définition d'une étiquette de texte
rb1.setText("Accueil");
// Un bouton radio avec le libellé spécifié
RadioButton rb2 = new RadioButton("Calendrier");
```

Vous pouvez explicitement sélectionner un bouton radio en utilisant la méthode `setSelected` et en spécifiant sa valeur comme `true`. Si vous avez besoin de vérifier si un bouton radio particulier a été sélectionné par un utilisateur, appliquez la méthode `isSelected`.

## Ajouter des boutons radio aux groupes

Étant donné que la classe `RadioButton` est une extension de la classe `Labeled`, vous pouvez spécifier non seulement une légende de texte, mais également une image. Utilisez la méthode `setGraphic` pour spécifier une image. [L'exemple 4-2](#) montre comment implémenter un bouton radio graphique dans votre application.

### Exemple 4-2 Création d'un bouton radio graphique

```
Image image = nouvelle Image(getClass().getResourceAsStream("ok.jpg"));
RadioButton rb = new RadioButton("Accepter");
rb.setGraphic(nouvelle ImageView(image));
```

## Ajouter des boutons radio aux groupes

Les boutons radio sont généralement utilisés dans un groupe pour présenter plusieurs options mutuellement exclusives. L'objet `ToggleGroup` fournit des références à tous les boutons radio qui lui sont associés et les gère de sorte qu'un seul des boutons radio puisse être sélectionné à la fois. [L'exemple 4-3](#) crée un groupe bascule, crée trois boutons radio, ajoute chaque bouton radio au groupe bascule et spécifie quel bouton doit être sélectionné au démarrage de l'application.

### Exemple 4-3 Création d'un groupe de boutons radio

```
groupe ToggleGroup final = new ToggleGroup();

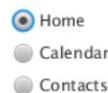
RadioButton rb1 = new RadioButton("Accueil");
rb1.setToggleGroup(groupe);
rb1.setSelected(vrai);

RadioButton rb2 = new RadioButton("Calendrier");
rb2.setToggleGroup(groupe);

RadioButton rb3 = new RadioButton("Contacts");
rb3.setToggleGroup(groupe);
```

Lorsque ces boutons radio sont disposés à l'aide des conteneurs de disposition et ajoutés au contenu de l'application, la sortie doit ressembler à la [Figure 4-2](#).

**Figure 4-2 Trois boutons radio combinés dans un groupe**



## Traitement des événements pour les boutons radio

Généralement, l'application exécute une action lorsqu'un des boutons radio du groupe est sélectionné. Passez en revue le fragment de code de l' [exemple 4-4](#) pour savoir comment modifier une icône en fonction du bouton radio sélectionné.

### Exemple 4-4 Action de traitement pour les boutons radio

```
ImageView image = new ImageView();
rb1.setUserData ("Accueil")
rb2.setUserData ("Calendrier");
```

```

rb3.setUserData("Contacts");

groupe ToggleGroup final = new ToggleGroup();
group.selectedToggleProperty().addListener(new ChangeListener<Toggle>(){
    public void modifié (ObservableValue<? étend Toggle> ov,
        Basculer old_toggle, Basculer new_toggle) {
            si (group.getSelectedToggle() != null) {
                image finale image = nouvelle image(
                    getClass().getResourceAsStream(
                        groupe.getSelectedToggle().getUserData().toString() +
                            ".jpg"
                    )
                );
                icon.setImage(image);
            }
        }
});

```

Les données utilisateur ont été attribuées pour chaque bouton radio. Le `ChangeListener<Toggle>` l'objet vérifie une bascule sélectionnée dans le groupe. Il utilise la méthode `getSelectedToggle` pour identifier le bouton radio actuellement sélectionné et extrait ses données utilisateur en appelant la méthode `getUserData`. Ensuite, les données utilisateur sont appliquées pour construire un nom de fichier image à charger.

Par exemple, lorsque `rb3` est sélectionné, la méthode `getSelectedToggle` renvoie "rb3" et la méthode `getUserData` renvoie "Contacts". Par conséquent, la méthode `getResourceAsStream` reçoit la valeur "Contacts.jpg". La sortie de l'application est illustrée à [la Figure 4–1](#).

## Demande de focus pour un bouton radio

Dans le groupe de boutons radio, le premier bouton a initialement le focus par défaut. Si vous appliquez la méthode `setSelected` au deuxième bouton radio du groupe, vous devez vous attendre au résultat illustré à la [Figure 4–3](#).

**Figure 4–3** Paramètres de mise au point par défaut



Le deuxième bouton radio est sélectionné et le premier bouton reste actif. Utilisez la fonction `requestFocus` pour modifier le focus, comme illustré dans l' [exemple 4–5](#).

### Exemple 4–5 Demande de focus pour le deuxième bouton radio

```

rb2.setSelected(true);
rb2.requestFocus();

```

Lorsqu'il est appliqué, ce code produit le résultat illustré à la [Figure 4–4](#).

Demande de focus pour un bouton radio

---

**Figure 4–4 Définition du focus pour le bouton radio sélectionné**



### Documentation relative à l'API

ÿ Bouton radio

ÿ Étiqueté

ÿ BasculerGroupe

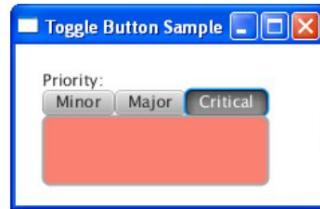
# 5

## 5Bouton à bascule

Dans ce chapitre, vous découvrirez la classe `ToggleButton`, un autre type de boutons disponible via l'API JavaFX.

Deux ou plusieurs boutons à bascule peuvent être combinés dans un groupe où un seul bouton à la fois peut être sélectionné, ou où aucune sélection n'est requise. La Figure 5–1 est une capture d'écran d'une application qui combine trois boutons bascule dans un groupe. L'application peint le rectangle avec une couleur spécifique en fonction du bouton bascule sélectionné.

**Figure 5–1 Trois boutons bascule**



### Création d'un bouton bascule

Vous pouvez créer un bouton bascule dans votre application en utilisant l'un des trois constructeurs de la classe `ToggleButton`, comme illustré dans l' [exemple 5–1](#).

#### Exemple 5–1 Création de boutons bascule

```
// Un bouton bascule sans légende ni icône
ToggleButton tb1 = new ToggleButton();
// Un bouton bascule avec une légende de texte
ToggleButton tb2 = new ToggleButton("Appuyez sur moi");
// Un bouton bascule avec une légende de texte et une icône
Image image = nouvelle Image(getClass().getResourceAsStream("icon.png"));
ToggleButton tb3 = new ToggleButton ("Press me", new ImageView(image));
```

La classe `ToggleButton` est une extension de la classe `Labeled`, vous pouvez donc spécifier une légende de texte, une image ou à la fois une image et du texte. Vous pouvez utiliser les méthodes `setText` et `setGraphic` de la classe `Labeled` pour spécifier le contenu textuel et graphique d'un bouton bascule.

Une fois que vous avez défini des boutons bascule dans votre code, vous pouvez les combiner dans un groupe et définir un comportement spécifique.

Ajout de boutons bascule à un groupe

## Ajout de boutons bascule à un groupe

L'implémentation de la classe ToggleButton est très similaire à l'implémentation de la classe RadioButton. Cependant, contrairement aux boutons radio, les boutons bascule d'un groupe bascule ne tentent pas de forcer la sélection d'au moins un bouton du groupe. Autrement dit, cliquer sur le bouton bascule sélectionné le désélectionne, cliquer sur le bouton radio sélectionné dans le groupe n'a aucun effet.

Prenez un moment pour étudier le fragment de code [Exemple 5–2](#).

### Exemple 5–2 Combinaison de boutons bascule dans un groupe

```
groupe ToggleGroup final = new ToggleGroup();
```

```
ToggleButton tb1 = new ToggleButton("Minor");
tb1.setToggleGroup(groupe);
tb1.setSelected(true);
```

```
ToggleButton tb2 = new ToggleButton("Major");
tb2.setToggleGroup(groupe);
```

```
ToggleButton tb3 = new ToggleButton("Critique");
tb3.setToggleGroup(groupe);
```

L'[exemple 5–2](#) crée trois boutons bascule et les ajoute au groupe bascule. La méthode setSelected est appelée pour le bouton bascule tb1 afin qu'il soit sélectionné au démarrage de l'application. Cependant, vous pouvez désélectionner le bouton bascule Mineur afin qu'aucun bouton bascule ne soit sélectionné dans le groupe au démarrage, comme illustré à la [Figure 5–2](#).

**Figure 5–2 Trois boutons bascule dans un groupe**



En règle générale, vous utilisez un groupe de boutons bascule pour attribuer un comportement spécifique à chaque bouton. La section suivante explique comment utiliser ces boutons bascule pour modifier la couleur d'un rectangle.

## Définition du comportement

La méthode setUserData héritée par la classe ToggleButton de la classe Node vous aide à associer toute option sélectionnée à une valeur particulière. Dans l'[exemple 5–3](#), les données utilisateur indiquent la couleur à utiliser pour peindre le rectangle.

### Exemple 5–3 Définition des données utilisateur pour les boutons bascule

```
tb1.setUserData(Color.LIGHTGREEN);
tb2.setUserData (Color.LIGHTBLUE);
tb3.setUserData(Color.SALMON);
```

```
Rectangle final rect = nouveau Rectangle(145, 50);
```

```
groupe ToggleGroup final = new ToggleGroup();
group.selectedToggleProperty().addListener(new ChangeListener<Toggle>(){
    public void modifié (ObservableValue<? étent Toggle> ov,
        Basculer basculer, Basculer new_toggle) {
```

```

        si (new_toggle == null)
            rect.setFill(Color.WHITE);
        autre
            rect.setFill(
                (Couleur) group.getSelectedToggle().getUserData()
            );
        }
    });
}

```

L'objet `ChangeListener<Toggle>` vérifie une bascule sélectionnée dans le groupe. Si aucun des boutons bascule n'est sélectionné, le rectangle est peint en blanc. Si l'un des boutons bascule est sélectionné, les appels consécutifs des méthodes `getSelectedToggle` et `getUserData` renvoient une couleur pour peindre le rectangle.

Par exemple, si un utilisateur sélectionne le bouton bascule `tb2`, l'appel `getSelectedToggle().getUserData()` renvoie `Color.LIGHTBLUE`. Le résultat est illustré à la [Figure 5–3](#).

**Figure 5–3 Utilisation des boutons bascule pour peindre un rectangle**



Consultez le fichier `ToggleButtonSample.java` pour examiner le code complet de l'application.

## Boutons à bascule de style

Vous pouvez améliorer cette application en appliquant des styles CSS aux boutons bascule. L'utilisation de CSS dans les applications JavaFX 2 est similaire à l'utilisation de CSS dans HTML, car chaque cas est basé sur la même spécification CSS. L'[exemple 5–4](#) utilise la méthode `setStyle` pour modifier les propriétés CSS `-fx-base` des boutons bascule.

### Exemple 5–4 Application de styles CSS à des boutons bascule

```

tb1.setStyle("-fx-base: vert clair;");  

tb2.setStyle("-fx-base: bleu clair;");  

tb3.setStyle("-fx-base: saumon;");  


```

Lorsqu'elles sont ajoutées au code de l'application, ces lignes modifient l'apparence visuelle des boutons bascule, comme illustré à la [Figure 5–4](#).

**Figure 5–4 Boutons bascule peints**



Boutons à bascule de style

---

Vous voudrez peut-être essayer d'autres propriétés CSS de la classe ToggleButton ou appliquer des animations, des transformations et des effets visuels disponibles dans l'API JavaFX.

**Documentation relative à l'API**

ÿ Bouton bascule

ÿ BasculerGroupe

# 6

## 6Case à cocher

Ce chapitre enseigne comment ajouter des cases à cocher à vos applications JavaFX.

Bien que les cases à cocher ressemblent à des boutons radio, elles ne peuvent pas être combinées en groupes de basculement pour permettre la sélection de plusieurs options à la fois. Voir les chapitres Bouton radio et Bouton bascule pour plus d'informations.

[La Figure 6–1](#) montre une capture d'écran d'une application dans laquelle trois cases à cocher sont utilisées pour activer ou désactiver des icônes dans une barre d'outils d'application.

**Figure 6–1 Exemple de case à cocher**



### Création de cases à cocher

[L'exemple 6–1](#) crée deux cases à cocher simples.

#### Exemple 6–1 Création de cases à cocher

```
//Une case à cocher sans légende
CheckBox cb1 = new CheckBox();
// Une case à cocher avec une légende de chaîne
CheckBox cb2 = new CheckBox("Deuxième");

cb1.setText("Premier");
cb1.setSelected(vrai);
```

Une fois que vous avez créé une case à cocher, vous pouvez la modifier en utilisant les méthodes disponibles via les API JavaFX. Dans l'[exemple 6–1](#), la méthode setText définit la légende de texte de la case à cocher c1. La méthode setSelected est définie sur true afin que la case cb1 soit cochée au démarrage de l'application.

### Définir un état

La case à cocher peut être définie ou non définie. Lorsqu'il est défini, vous pouvez le sélectionner ou le désélectionner. Cependant, lorsque la case à cocher n'est pas définie, elle ne peut pas être sélectionnée ou désélectionnée. Utilisez une combinaison de setSelected et setIndeterminate

méthodes de la classe CheckBox pour spécifier l'état de la case à cocher. [Le Tableau 6–1](#) montre trois états d'une case à cocher en fonction de ses propriétés INDÉTERMINÉE et SÉLECTIONNÉE.

**Tableau 6–1 États d'une case à cocher**

Valeurs de propriété	Apparence de la case à cocher
INDÉTERMINÉ = faux	<input type="checkbox"/> I agree
SÉLECTIONNÉ = faux	<input checked="" type="checkbox"/> I agree
INDÉTERMINÉ = vrai	<input type="checkbox"/> I agree
SÉLECTIONNÉ = vrai/faux	<input checked="" type="checkbox"/> I agree

Vous devrez peut-être activer trois états pour les cases à cocher dans votre application lorsqu'elles représentent des éléments d'interface utilisateur qui peuvent être dans des états mixtes, par exemple, "Oui", "Non", "Non applicable". La propriété allowIndeterminate de l'objet CheckBox détermine si la case à cocher doit parcourir les trois états: sélectionnée, désélectionnée et non définie. Si la variable est vraie, la commande passera par les trois états. S'il est faux, le contrôle passera par les états sélectionnés et désélectionnés. L'application décrite dans la section suivante construit trois cases à cocher et n'active que deux états pour celles-ci.

## Définition du comportement

Le fragment de code de l' [exemple 6–2](#) crée trois cases à cocher, de sorte que si une case est cochée, l'icône correspondante apparaît dans une barre d'outils.

### Exemple 6–2 Définition du comportement des cases à cocher

```
final String[] names = new String[]{"Security", "Project", "Chart"};
Image finale[] images = nouvelle Image[noms.longueur];
icônes ImageView finales [] = new ImageView [names.length];
CheckBox final[] cbs = new CheckBox[noms.longueur];

for (int i = 0; i < noms.longueur; i++) {
    Image finale image = images[i] =
        nouvelle Image(getClass().getResourceAsStream(names[i] + ".png"));
    icône ImageView finale = icônes[i] = new ImageView();
    CheckBox final cb = cbs[i] = new CheckBox(names[i]);
    cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
        public void changé (ObservableValue<? étend Boolean> ov,
                           Ancienne_val booléenne, nouvelle_val booléenne) {
            icon.setImage(new_val ? image : null);
        }
    });
}
```

Le tableau de noms utilise une boucle for pour créer un tableau de cases à cocher et un tableau correspondant d'icônes. Par exemple, cbs[0], la première case à cocher, se voit attribuer la légende de texte "Sécurité". En même temps, image[0] reçoit "Security.png" comme nom de fichier pour la méthode getResourceStream lorsqu'une image pour la première icône est créée. Si une case particulière est cochée, l'image correspondante est attribuée à l'icône. Si une case est décochée, l'icône reçoit une image nulle et l'icône n'est pas rendue.

La [Figure 6–2](#) montre une application lorsque les cases Sécurité et Graphique sont cochées et que la case Projet est décochée.

**Figure 6–2 Application de case à cocher en action**



## Styliser une case à cocher

Les cases à cocher de la [Figure 6–2](#) ont l'apparence par défaut de la classe CheckBox.

Vous pouvez modifier l'apparence d'une case à cocher à l'aide de la méthode `setStyle`, comme illustré dans l'[exemple 6–3](#).

### Exemple 6–3 Stylisation d'une case à cocher

```
cb1.setStyle(
    "-fx-border-color: bleu clair; " + "-fx-font-
    size: 20; " + "-fx-border-insets: -5; " + "-fx-
    border-radius: 5; " + "-fx-border-style:
    pointillé; " + "-fx-border-width: 2;"

);
```

Le nouveau style comprend une bordure bleu clair en pointillés et une taille de police accrue pour sa légende de texte. La [Figure 6–3](#) montre la case à cocher cb1 avec ce style appliqué.

**Figure 6–3 Case à cocher stylisée**



Pour définir un style spécifique pour toutes les cases à cocher de votre application, utilisez la procédure suivante:

- ÿ Créez un fichier .css.

- ÿ Créez la classe CSS de case à cocher dans le fichier .css.

- ÿ Définissez tous les styles requis dans la classe CSS de la case à cocher. ÿ

Dans votre application JavaFX, activez la feuille de style en utilisant le `setStyleClass` méthode.

### Documentation relative à l'API

- ÿ Case à cocher

- ÿ Spécification CSS JavaFX

Styliser une case à cocher

---

sept

## 7Boîte à choix

Ce chapitre décrit les boîtes de choix, les contrôles de l'interface utilisateur qui permettent de sélectionner rapidement entre quelques options.

Utilisez la classe ChoiceBox pour ajouter des boîtes de choix à vos applications JavaFX. Sa mise en œuvre simple est illustrée à la [Figure 7–1](#).

**Figure 7–1 Création d'une boîte de sélection avec trois éléments**



### Création d'une boîte de choix

L'[exemple 7–1](#) crée une boîte de choix avec trois éléments.

#### Exemple 7–1 Création d'une boîte de choix

```
ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "Premier Deuxième Troisième")
);
```

L'[exemple 7–1](#) montre une liste d'éléments créés et remplis dans un constructeur de la classe ChoiceBox. Les éléments de la liste sont spécifiés à l'aide d'un tableau observable.

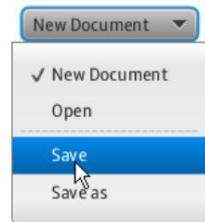
Vous pouvez également utiliser un constructeur vide de la classe et définir les éléments de la liste à l'aide de la méthode setItems présentée dans l' [exemple 7–2](#).

#### Exemple 7–2 Zone de choix avec des éléments de texte et un séparateur

```
ChoiceBox cb = new ChoiceBox();
cb.setItems(FXCollections.observableArrayList(
    "Nouveau document", "Ouvrir",
    nouveau séparateur(), "Enregistrer", "Enregistrer sous")
);
```

Notez qu'une boîte de choix peut contenir non seulement des éléments de texte, mais également d'autres objets. Une commande Séparateur est utilisée dans l' [exemple 7–2](#) pour séparer les éléments. Lorsque ce fragment de code est intégré à l'application, il produit la sortie illustrée à la [Figure 7–2](#).

Figure 7–2 Menu créé à l'aide d'une boîte de sélection



Dans les applications réelles, les boîtes de choix sont utilisées pour créer des listes à choix multiples.

## Définition du comportement d'une boîte de choix

L'application illustrée à la [Figure 7–3](#) propose une boîte à choix multiples avec cinq options. Lorsqu'une langue particulière est sélectionnée, le message d'accueil correspondant est rendu.

Figure 7–3 Liste à choix multiples



[La Figure 7–4](#) fournit un fragment de code pour illustrer comment un élément sélectionné dans la boîte de choix définit le message d'accueil à rendre.

Figure 7–4 Sélection d'un élément de la boîte de sélection

```
final String[] greetings = new String[]{"Hello", "Hola", "Привет", "你好",
                                         "こんにちは"};
final ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "English", "Español", "Русский", "简体中文", "日本語"
));
cb.getSelectionModel().selectedIndexProperty().addListener(new
    ChangeListener<Number>() {
    public void changed(ObservableValue ov,
                        Number value, Number new_value) {
        label.setText(greetings[new_value.intValue()]);
    }
});
```

Un objet `ChangeListener<Number>` détecte l'index de l'élément actuellement sélectionné par des appels consécutifs de `getSelectionModel` et `selectedIndexProperty` méthodes. La méthode `getSelectionModel` renvoie l'élément sélectionné et la méthode `selectedIndexProperty` renvoie la propriété `SELECTED_INDEX` de la boîte de choix `cb`. Par conséquent, la valeur entière en tant qu'index définit un élément du tableau de salutations et spécifie une valeur de texte de chaîne pour l'étiquette. Si, par exemple, un utilisateur sélectionne le deuxième élément, qui correspond à l'espagnol, `SELECTED_INDEX` est égal à 1 et "Hola" est sélectionné dans le tableau des salutations. Ainsi, l'étiquette rend "Hola".

Vous pouvez rendre le contrôle ChoiceBox plus informatif en lui attribuant une info-bulle. Une info-bulle est un contrôle d'interface utilisateur disponible dans le package javafx.scene.control. Une info-bulle peut être appliquée à n'importe lequel des contrôles de l'interface utilisateur JavaFX.

## Application d'une info-bulle

La classe Tooltip fournit une info-bulle préfabriquée qui peut être facilement appliquée à une boîte de choix (ou à tout autre contrôle) en appelant la méthode setTooltip illustrée dans l'[exemple 7–3](#).

### Exemple 7–3 Ajout d'une info-bulle à une boîte de choix

```
cb.setTooltip(new Tooltip("Sélectionnez la langue"));
```

Généralement, un utilisateur définit le texte de l'info-bulle dans un constructeur de l'info-bulle classer. Toutefois, si la logique de votre application nécessite que l'interface utilisateur définisse le texte de manière dynamique, vous pouvez appliquer une info-bulle à l'aide d'un constructeur vide, puis lui affecter le texte à l'aide de la méthode setText.

Une fois l'info-bulle appliquée à la boîte de choix cb, un utilisateur qui positionne le curseur sur la boîte de choix voit l'image illustrée à la [Figure 7–5](#).

**Figure 7–5 Boîte de sélection avec l'info-bulle appliquée**



Pour améliorer encore votre application, vous pouvez styliser la boîte de choix avec les propriétés CSS ou appliquer des effets visuels ou des transformations.

#### Documentation relative à l'API

ÿ Boîte de choix

ÿ Info-bulle



# 8

## 8Champ de texte

Ce chapitre traite des fonctionnalités du contrôle de champ de texte.

La classe `TextField` implémente un contrôle d'interface utilisateur qui accepte et affiche la saisie de texte. Il fournit des capacités pour recevoir une entrée de texte d'un utilisateur. Avec un autre contrôle de saisie de texte, `PasswordField`, cette classe étend la classe `TextInput`, une super classe pour tous les contrôles de texte disponibles via l'API JavaFX.

[La Figure 8–1](#) montre un champ de texte typique avec une étiquette.

**Figure 8–1 Étiquette et champ de texte**

Name:

### Création d'un champ de texte

Dans l' [exemple 8–1](#), un champ de texte est utilisé en combinaison avec une étiquette pour indiquer le type de contenu qui doit être saisi dans le champ.

#### Exemple 8–1 Création d'un champ de texte

```
Étiquette étiquette1 = nouvelle étiquette("Nom:");
TextField textField = nouveau TextField ();
HBox hb = new HBox();
hb.getChildren().addAll(label1, textField);
hb.setSpacing(10);
```

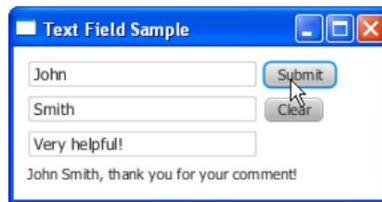
Vous pouvez créer un champ de texte vide comme illustré dans l' [exemple 8–1](#) ou un champ de texte contenant des données de texte particulières. Pour créer un champ de texte avec le texte prédéfini, utilisez le constructeur suivant de la classe `TextField` : `TextField("Hello World!")`. Vous pouvez obtenir la valeur d'un champ texte à tout moment en appelant la méthode `getText`.

Vous pouvez appliquer la méthode `setPrefColumnCount` de la classe `TextInput` pour définir la taille du champ de texte, définie comme le nombre maximum de caractères qu'il peut afficher à la fois.

### Construire l'interface utilisateur avec des champs de texte

Généralement, les objets `TextField` sont utilisés dans les formulaires pour créer plusieurs champs de texte.

L'application de la [Figure 8–2](#) affiche trois champs de texte et traite les données qu'un utilisateur y entre.

**Figure 8–2 Exemple d'application TextField**

Le fragment de code de l' [exemple 8–2](#) crée les trois champs de texte et les deux boutons, et les ajoute à la scène de l'application à l'aide du conteneur GridPane. Ce conteneur est particulièrement pratique lorsque vous devez implémenter une disposition flexible pour vos contrôles d'interface utilisateur.

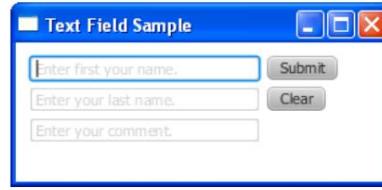
### Exemple 8–2 Ajout de champs de texte à l'application

```
//Création d'un conteneur GridPane
GridPane grid = new GridPane();
grid.setPadding(nouveaux inserts(10, 10, 10, 10));
grille.setVgap(5); grille.setHgap(5); //Définition du champ de
texte Nom final TextField name = new TextField();
name.setPromptText("Entrez votre prénom.");
nom.setPrefColumnCount(10);

nom.getText();
GridPane.setConstraints(nom, 0, 0);
grille.getChildren().add(nom); //Définition
du champ de texte Nom de famille final TextField
lastName = new TextField(); lastName.setPromptText("Entrez
votre nom de famille."); GridPane.setConstraints(lastName, 0, 1);
grille.getChildren().add(lastName); //Définition du champ de texte
Commentaire final TextField comment = new TextField();
comment.setPrefColumnCount(15); comment.setPromptText("Entrez
votre commentaire."); GridPane.setConstraints(commentaire, 0, 2);
grille.getChildren().add(commentaire); //Définition du bouton Submit
Button submit = new Button("Submit");
GridPane.setConstraints(soumettre, 1, 0);
grille.getChildren().add(soumettre); //Définition du bouton Clear
Button clear = new Button("Clear"); GridPane.setConstraints(clear,
1, 1);

grille.getChildren().add(clear);
```

Prenez un moment pour étudier le fragment de code. Les champs de texte name, lastName et comment sont créés à l'aide de constructeurs vides de la classe TextField. Contrairement à l' [exemple 8–1](#), les étiquettes n'accompagnent pas les champs de texte dans ce fragment de code. Au lieu de cela, les légendes des invites informent les utilisateurs du type de données à saisir dans les champs de texte. La méthode setPromptText définit la chaîne qui apparaît dans le champ de texte au démarrage de l'application. Lorsque l' [Exemple 8–2](#) est ajouté à l'application, il produit la sortie illustrée à la [Figure 8–3](#).

**Figure 8–3 Trois champs de texte avec les messages d'invite**

La différence entre le texte de l'invite et le texte saisi dans le champ de texte est que le texte de l'invite ne peut pas être obtenu via la méthode `getText`.

Dans les applications réelles, les données saisies dans les champs de texte sont traitées conformément à la logique d'une application telle que requise par une tâche métier spécifique. La section suivante explique comment utiliser les champs de texte pour évaluer les données saisies et générer une réponse à un utilisateur.

### Traitement des données de champ de texte

Comme mentionné précédemment, les données de texte saisies par un utilisateur dans les champs de texte peuvent être obtenues par la méthode `getText` de la classe `TextInput`.

Étudiez l' [exemple 8–3](#) pour apprendre à traiter les données de l'objet `TextField`.

#### **Exemple 8–3 Définition d'actions pour les boutons Soumettre et Effacer**

```
//Ajout d'un Label final
Label label = new Label();
GridPane.setConstraints(étiquette, 0, 3);
GridPane.setColumnSpan(étiquette, 2);
grille.getChildren().add(étiquette);

//Définition d'une action pour le bouton Soumettre
submit.setOnAction(new EventHandler<ActionEvent>() {

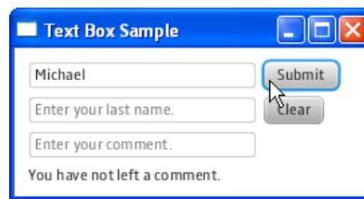
    @Passer outre
    public void handle(ActionEvent e) {
        if ((comment.getText() != null && !comment.getText().isEmpty()) && (lastName.getText() != null && !lastName.getText().isEmpty())) {
            .. ..
            label.setText(nom.getText() +
                + "merci pour votre commentaire !");
        } else {
            label.setText("Vous n'avez pas laissé de commentaire.");
        }
    }
});

//Définition d'une action pour le bouton Effacer
clear.setOnAction(new EventHandler<ActionEvent>() {

    @Passer outre
    public void handle(ActionEvent e) {
        name.clear();
        nom.clear(); comment.clear(); label.setText(null);
    }
});
```

Le contrôle Label ajouté au conteneur GridPane restitue la réponse d'une application aux utilisateurs. Lorsqu'un utilisateur clique sur le bouton Soumettre, la méthode setOnAction vérifie le champ de texte de commentaire. S'il contient une chaîne non vide, un message de remerciement est rendu. Sinon, l'application avertit l'utilisateur que le message de commentaire n'a pas encore été laissé, comme illustré à la [Figure 8–4](#).

Figure 8–4 Le champ de texte de commentaire laissé vide



Lorsqu'un utilisateur clique sur le bouton Effacer, le contenu est effacé dans les trois champs de texte.

Passez en revue certaines méthodes utiles que vous pouvez utiliser avec les champs de texte.

ÿ copy() – transfère la plage actuellement sélectionnée dans le texte vers le presse-papiers, laissant la sélection actuelle.

ÿ cut() – transfère la plage actuellement sélectionnée dans le texte vers le presse-papiers, en supprimant la sélection actuelle.

ÿ paste() – transfère le contenu du presse-papiers dans ce texte, remplaçant la sélection actuelle.

#### Documentation relative à l'API

ÿ Champ de texte

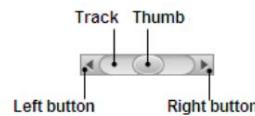
ÿ TextInputControl

## 9Barre de défilement

Ce chapitre explique comment créer des volets défilants à l'aide du contrôle de la barre de défilement.

La classe ScrollBar vous permet de créer des volets et des vues défilants dans votre application. La [Figure 9–1](#) montre les trois zones d'une barre de défilement : le pouce, les boutons droit et gauche (ou les boutons bas et haut) et la piste.

**Figure 9–1 Éléments de la barre de défilement**



### Création d'une barre de défilement

Prenez quelques instants pour examiner le fragment de code de l' [exemple 9–1](#).

#### Exemple 9–1 Barre de défilement simple

```
ScrollBar sc = new ScrollBar();
sc.setMin(0);
sc.setMax(100);
sc.setValue(50);
```

Les méthodes setMin et setMax définissent les valeurs minimum et maximum représentées par la barre de défilement. Lorsqu'un utilisateur déplace le pouce, la valeur de la barre de défilement change. Dans l' [exemple 9–1](#), la valeur est égale à 50, donc lorsque l'application démarre, le pouce se trouve au centre de la barre de défilement. Par défaut, la barre de défilement est orientée horizontalement. Cependant, vous pouvez définir l'orientation verticale à l'aide de la méthode setOrientation.

L'utilisateur peut cliquer sur le bouton gauche ou droit (bouton bas ou haut pour l'orientation verticale) pour faire défiler par incrément d'unité. La propriété UNIT\_INCREMENT spécifie de combien la barre de défilement est ajustée lorsqu'un bouton est cliqué. Une autre option consiste à cliquer dans la piste par incrément de bloc. La propriété BLOCK\_INCREMENT définit la valeur par laquelle la barre de défilement est ajustée lorsque la piste de la barre est cliquée.

Dans votre application, vous pouvez utiliser l'une des nombreuses barres de défilement pour faire défiler le contenu graphique qui dépasse les limites de l'espace disponible.

## Utilisation d'une barre de défilement dans votre application

Examinez la barre de défilement en action. L'application présentée dans l'[exemple 9–2](#) implémente une scène déroulante pour afficher les images. La tâche de cette application est de permettre aux utilisateurs de visualiser le contenu de la boîte verticale, qui est plus longue que la hauteur de la scène.

### Exemple 9–2 Défilement de plusieurs images

```
importer javafx.application.Application; importer
javafx.beans.value.ChangeListener<?>; importez
javafx.beans.value.ObservableValue<?>; importer
javafx.geometry.Orientation<?>; importer javafx.scene.Group<?>;
importer javafx.scene.Scene<?>; importer
javafx.scene.control.ScrollBar<?>; importer
javafx.scene.effect.DropShadow<?>; importer
javafx.scene.image.Image<?>; importer
javafx.scene.image.ImageView<?>; importer
javafx.scene.layout.VBox<?>; importer javafx.scene.paint.Color<?>;
importer javafx.stage.Stage<?>;
```

```
la classe publique principale étend l'application {

    barre de défilement finale sc = nouvelle barre de
    défilement(); Images finales Image[] = nouvelle
    Image[5]jy; photos ImageView[] finales = nouvelle
    ImageView[5]jy; VBox finale vb = nouvelle VBox();
    DropShadow shadow = new DropShadow();

    @Passer outre
    public void start(Stage stage) { Group root =
        new Group(); Scene scene = new
        Scene(root, 180, 180); scene.setFill(Color.BLACK);

        stage.setScene(scene);
        stage.setTitle("Barre de défilement");
        root.getChildren().addAll(vb, sc);

        shadow.setColor(Color.GREY);
        shadow.setOffsetX(2);
        shadow.setOffsetY(2);

        vb.setLayoutX(5);
        vb.setSpacing(10);

        sc.setLayoutX(scene.getWidth()-sc.getWidth()); sc.setMin(0);
        sc.setOrientation(Orientation.VERTICAL); sc.setPrefHeight(180);
        sc.setMax(360);

    pour (int je = 0; je < 5; je++) {
        Image finale image = images[i] =
            nouvelle Image(getClass().getResourceAsStream(
                "fw" + (i + 1) + ".jpg")
            );
        ImageView final pic = pics[i] = new
        ImageView(images[i]); pic.setEffect(ombre);
```

```

        vb.getChildren().add(pics[i]);
    }

    sc.valueProperty().addListener(new ChangeListener<Number>() {
        public void modifié (ObservableValue<? étend le nombre> ov,
            Numéro ancien_val, Numéro nouveau_val) {
            vb.setLayoutY(-new_val.doubleValue());
        }
    });

    spectacle();
}

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

```

Les premières lignes du code ajoutent une boîte verticale avec les images et une barre de défilement au scène.

La coordonnée Y de la boîte verticale change lorsque la propriété VALUE de la barre de défilement est modifiée, de sorte qu'à chaque fois que le pouce est déplacé ou que l'on clique sur un bouton ou sur la piste, la boîte verticale se déplace, comme illustré dans l' [exemple 9–3](#) .

#### **Exemple 9–3 Implémentation du défilement de la boîte verticale**

```

sc.valueProperty().addListener(new ChangeListener<Number>() {
    public void modifié (ObservableValue<? étend le nombre> ov,
        Numéro ancien_val, Numéro nouveau_val) {
            vb.setLayoutY(-new_val.doubleValue());
        }
    });
}
}

```

La compilation et l'exécution de cette application produisent la sortie illustrée à la [Figure 9–2](#).

**Figure 9–2 Exemple de barre de défilement**



Cette application montre une utilisation typique de la classe ScrollBar. Vous pouvez également personnaliser cette classe pour créer une zone de défilement dans une scène. Comme pour chaque contrôle de l'interface utilisateur et chaque nœud, la barre de défilement peut être stylisée pour changer son apparence par rapport à l'implémentation par défaut.

#### **Documentation relative à l'API**

ÿ Barre de défilement

ÿ Spécification CSS JavaFX

Utilisation d'une barre de défilement dans votre application

---

# dix

## 10Volet de défilement

Dans ce chapitre, vous apprendrez à créer des volets de défilement dans vos applications JavaFX.

Les volets de défilement fournissent une vue déroulante des éléments de l'interface utilisateur. Ce contrôle permet à l'utilisateur de faire défiler le contenu en déplaçant la fenêtre d'affichage ou en utilisant des barres de défilement. Un volet de défilement avec les paramètres par défaut et l'image ajoutée est illustré à la [Figure 10–1](#).

**Figure 10–1 Volet de défilement**



### Création d'un volet de défilement

[L'exemple 10–1](#) montre comment créer ce volet de défilement dans votre application.

#### Exemple 10–1 Utilisation d'un volet de défilement pour afficher une image

```
Image roses = new Image(getClass().getResourceAsStream("roses.jpg"));
ScrollPane sp = new ScrollPane();
sp.setContent(new ImageView(roses));
```

La méthode setContent définit le nœud utilisé comme contenu de ce volet de défilement. Vous ne pouvez spécifier qu'un seul nœud. Pour créer une vue de défilement avec plusieurs composants, utilisez des conteneurs de disposition ou la classe Group. Vous pouvez également spécifier le vrai valeur pour la méthode setPannable pour prévisualiser l'image en cliquant dessus et en déplaçant le curseur de la souris. La position des barres de défilement change en conséquence.

### Définition de la stratégie de barre de défilement pour un volet de défilement

La classe ScrollPane fournit une stratégie pour déterminer quand afficher les barres de défilement: toujours, jamais ou uniquement lorsqu'elles sont nécessaires. Utilisez les méthodes setHbarPolicy et setVbarPolicy pour spécifier la stratégie de barre de défilement pour les barres de défilement horizontales et verticales respectivement. Ainsi, dans l'[exemple 10–2](#), la barre de défilement verticale apparaîtra, mais pas la barre de défilement horizontale.

#### Exemple 10–2 Définition des stratégies de barre de défilement horizontale et verticale

```
sp.setHbarPolicy(ScrollBarPolicy.NEVER);
sp.setVbarPolicy(ScrollBarPolicy.ALWAYS);
```

Par conséquent, vous ne pouvez faire défiler l'image que verticalement, comme illustré à la [Figure 10–2](#).

**Figure 10–2 Désactivation de la barre de défilement horizontale**

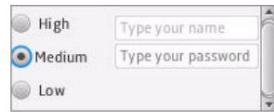


### Redimensionnement des composants dans le volet de défilement

Lors de la conception d'une interface utilisateur, vous devrez peut-être redimensionner les composants afin qu'ils correspondent à la largeur ou à la hauteur du volet de défilement. Définissez la méthode `setFitToWidth` ou `setFitToHeight` sur true pour correspondre à une dimension particulière.

Le volet de défilement illustré à la [Figure 10–3](#) contient des boutons radio, une zone de texte et une zone de mot de passe. La taille du contenu dépasse la taille prédéfinie du volet de défilement et une barre de défilement verticale apparaît. Cependant, étant donné que la méthode `setFitToWidth` définit true pour le volet de défilement, le contenu se réduit en largeur et ne défile jamais horizontalement.

**Figure 10–3 Ajustement de la largeur du volet de défilement**



Par défaut, les propriétés `FIT_TO_WIDTH` et `FIT_TO_HEIGHT` sont fausses et le contenu redimensionnable conserve sa taille d'origine. Si vous supprimez les méthodes `setFitToWidth` du code de cette application, vous verrez la sortie illustrée à la [Figure 10–4](#).

**Figure 10–4 Propriétés par défaut pour l'ajustement du contenu**



La classe `ScrollPane` vous permet de récupérer et de définir les valeurs actuelles, minimales et maximales du contenu dans les directions horizontale et verticale. Apprenez à les utiliser dans vos applications.

### Exemple d'application avec un volet de défilement

L'[exemple 10–3](#) utilise un volet de défilement pour afficher une boîte verticale avec des images. La `VVALEUR` La propriété de la classe `ScrollPane` permet d'identifier l'image actuellement affichée et de restituer le nom du fichier image.

#### Exemple 10–3 Utilisation d'un volet de défilement pour afficher des images

paquet scrollpanesample;

```
importer javafx.application.Application;
importer javafx.beans.value.ChangeListener;
```

```

importez javafx.beans.value.ObservableValue; importer
javafx.scene.Scene; importer javafx.scene.control.Label; importer
javafx.scene.control.ScrollPane; importer javafx.scene.image.Image;
importer javafx.scene.image.ImageView; importer
javafx.scene.layout.Priority; importer javafx.scene.layout.VBox;
importer javafx.stage.Stage;

la classe publique principale étend l'application {

dernier ScrollPane sp = new ScrollPane(); Images finales
Image[] = nouvelle Image[5]; photos ImageView[] finales =
nouvelle ImageView[5]; VBox finale vb = nouvelle VBox(); Étiquette
finale nom_fichier = nouvelle étiquette();

chaîne finale [] imageNames = nouvelle chaîne [] {"fw1.jpg", "fw2.jpg",
"fw3.jpg", "fw4.jpg", "fw5.jpg"}};

@Passer outre
public void start(Stage stage) {
    Boîte VBox = nouvelle VBox();
    Scene scene = new Scene(box, 180, 180); stage.setScene(scene);
    stage.setTitle("Scroll Pane"); box.getChildren().addAll(sp,
    fileName); VBox.setVgrow(sp, Priority.ALWAYS);

    fileName.setLayoutX(30);
    fileName.setLayoutY(160);

    for (int i = 0; i < 5; i++) { images[i] = new
        Image(getClass().getResourceAsStream(imageNames[i])); pics[i] = new ImageView(images[i]);
        pics[i].setFitWidth(100); pics[i].setPreserveRatio(true); vb.getChildren().add(pics[i]);

    }

    sp.setMax(440);
    sp.setPrefSize(115, 150);
    sp.setContent(vb); sp.vvalueProperty
    (. /100);

    }

});
spectacle();
}

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

}
}

```

La compilation et l'exécution de cette application produisent la fenêtre illustrée à la [Figure 10–5](#).

Exemple d'application avec un volet de défilement

**Figure 10–5 Défilement des images**



La valeur maximale de la barre de défilement verticale est égale à la hauteur de la boîte verticale.  
Le fragment de code illustré dans l' [exemple 10–4](#) restitue le nom du fichier image actuellement affiché.

**Exemple 10–4 Suivi de la modification de la valeur verticale du volet de défilement**

```
sp.vvalueProperty().addListener(new ChangeListener<Number>() {  
    public void modifié (ObservableValue<? étend le nombre> ov,  
        Numéro ancien_val, Numéro nouveau_val) {  
        fileName.setText(imageNames[(new_val.intValue() - 1)/100]);  
    }  
});
```

L'objet ImageView limite la hauteur de l'image à 100 pixels. Par conséquent, lorsque new\_val.intValue() - 1 est divisé par 100, le résultat donne l'index de l'image courante dans le tableau imageNames.

Dans votre application, vous pouvez également faire varier les valeurs minimales et maximales des barres de défilement verticales et horizontales, mettant ainsi à jour dynamiquement votre interface utilisateur.

**Documentation relative à l'API**

- ÿ Panneau de défilement
- ÿ Barre de défilement

# 11

## 11Vue en liste

Dans ce chapitre, vous apprendrez à créer des listes dans vos applications JavaFX.

La classe ListView représente une liste déroulante d'éléments. La [Figure 11–1](#) montre la liste des types d'hébergement disponibles dans un système de réservation d'hôtel.

**Figure 11–1 Affichage de liste simple**



Vous pouvez remplir la liste en définissant ses éléments avec la méthode `setItems`. Vous pouvez également créer une vue pour les éléments de la liste en appliquant la méthode `setCellFactory`.

### Création d'une vue de liste

Le fragment de code de l' [exemple 11–1](#) implémente la liste avec les éléments String illustrés à la [figure 11–1](#).

#### Exemple 11–1 Création d'un contrôle de vue de liste

```
ListView<String> list = new ListView<String>();
ObservableList<String> items = FXCollections.observableArrayList(
    "Simple", "Double", "Suite", "Application Famille");
list.setItems(articles);
```

Pour modifier la taille et la hauteur du contrôle de vue de liste, utilisez les méthodes `setPrefHeight` et `setPrefWidth`. L' [exemple 11–2](#) contraint la liste verticale à 100 pixels de large sur 70 pixels de haut, ce qui donne la liste illustrée à la [figure 11–2](#).

#### Exemple 11–2 Définition de la hauteur et de la largeur d'un affichage de liste

```
list.setPrefWidth(100);
list.setPrefHeight(70);
```

**Figure 11–2 Liste verticale redimensionnée**

Vous pouvez orienter un objet ListView horizontalement en définissant la propriété orientation sur Orientation.HORIZONTAL. Cela peut être fait comme suit : list.setOrientation(Orientation.HORIZONTAL). La liste horizontale avec les mêmes éléments qu'à la [Figure 11–1](#) est illustrée à la [Figure 11–3](#).

**Figure 11–3 Contrôle de la vue de liste horizontale**

À tout moment, vous pouvez suivre la sélection et le focus de l'objet ListView avec les classes SelectionModel et FocusModel. Pour obtenir l'état actuel de chaque élément, utilisez une combinaison des méthodes suivantes:

- ÿ `getSelectionModel().getSelectedIndex()` – Renvoie l'index des éléments actuellement sélectionnés en mode de sélection unique
- ÿ `getSelectionModel().getSelectedItem()` – Renvoie le modèle actuellement sélectionné Objet
- ÿ `getFocusModel().getFocusedIndex()` – Renvoie l'index du modèle actuellement élément ciblé
- ÿ `getFocusModel().getFocusedItem()` – Renvoie l'élément actuellement ciblé

Le SelectionModel par défaut utilisé lors de l'instanciation d'un ListView est une implémentation de la classe abstraite MultipleSelectionModel. Cependant, la valeur par défaut de la propriété selectionMode est SelectionMode.SINGLE. Pour activer la sélection multiple dans une instance ListView par défaut, utilisez la séquence d'appels suivante:

```
listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

Notez également que MultipleSelectionModel a les propriétés `selectedItems` et `selectedIndices`, qui sont toutes deux des listes observables qui peuvent être surveillées pour détecter toute sélection multiple.

## Remplir une vue de liste avec des données

L'[exemple 11–1](#) montre la manière la plus simple de remplir une vue de liste. Pour enrichir votre liste, vous pouvez ajouter des données de différents types en utilisant les extensions spécifiques du ListCell classe, comme CheckBoxListCell, ChoiceBoxListCell, ComboBoxListCell et TextFieldListCell. Ces classes apportent des fonctionnalités supplémentaires à la cellule de liste de base. L'implémentation de fabrique de cellules pour de telles classes permet aux développeurs de modifier les données directement dans la vue de liste.

Par exemple, le contenu d'une cellule de liste n'est pas modifiable par défaut. Cependant, la classe ComboBoxListCell dessine une zone de liste déroulante à l'intérieur de la cellule de liste. Cette modification permet aux utilisateurs de créer une liste de noms en les sélectionnant dans une liste déroulante, comme illustré dans l'[exemple 11–3](#).

**Exemple 11–3 Ajout d'éléments ComboBoxListCell à une vue de liste**

```
importer javafx.application.Application; importer
javafx.collections.FXCollections; importer
javafx.collections.ObservableList; importer
javafx.scene.Scene; importer javafx.scene.control.*;
importer javafx.scene.control.cell.ComboBoxListCell;
importer javafx.scene.layout.StackPane; importer javafx.stage.Stage;
```

```
la classe publique ListViewSample étend l'application {
```

```
    public static final ObservableList names =
        FXCollections.observableArrayList(); public static final
    ObservableList data = FXCollections.observableArrayList();
```

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

```
    }
```

```
    @Passer autre
```

```
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Exemple de vue de liste");
```

```
        ListView final listView = new ListView(data); listView.setPrefSize(200,
        250);
        listView.setEditable(true);
```

```
        noms.addAll(
            "Adam", "Alex", "Alfred", "Albert",
            "Brenda", "Connie", "Derek", "Donny",
            "Lynne", "Myrtle", "Rose", "Rodolphe",
            "Tony", "Trudy", "Williams", "Zach"
        );
```

```
        for (int i = 0; i < 18; i++) { data.add("anonym");
```

```
    }
```

```
    listView.setItems(data);
listView.setCellFactory(ComboBoxListCell.forListView(names));
```

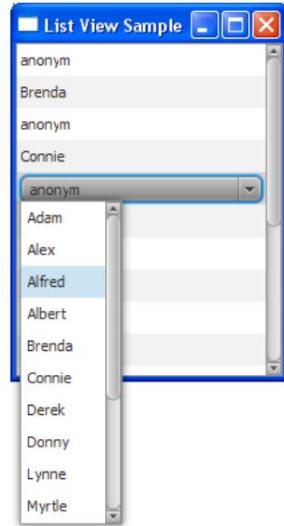
  

```
    StackPane racine = nouveau StackPane();
    root.getChildren().add(listView);
    primaryStage.setScene(nouvelle scène(racine, 200, 250));
    primaryStage.show();
}
```

  
}

La ligne en gras dans l'exemple appelle la méthode `setCellFactory` pour redéfinir l'implémentation de la cellule de liste. Lorsque vous compilez et exécutez cet exemple, il produit la fenêtre d'application illustrée à la [Figure 11–4](#).

Figure 11–4 Vue Liste avec les cellules de la zone de liste déroulante



Non seulement le mécanisme de fabrique de cellules vous permet d'appliquer les implémentations alternatives des cellules de la liste, mais il peut également vous aider à personnaliser totalement l'apparence des cellules.

## Personnalisation du contenu d'une vue de liste

Étudiez l'application suivante pour savoir comment générer les éléments de la liste à l'aide de la fabrique de cellules. L'application présentée dans l'[exemple 11–4](#) crée une liste de modèles de couleur.

### Exemple 11–4 Création d'une fabrique de cellules

```
importer javafx.application.Application; importer
javafx.collections.FXCollections; importer
javafx.collections.ObservableList; importer
javafx.scene.Scene; importer javafx.scene.control.ListView;
importer javafx.scene.control.ListCell; importer
javafx.scene.layout.Priority; importer
javafx.scene.layout.VBox; importer javafx.scene.paint.Color;
importer javafx.scene.shape.Rectangle; importer
javafx.stage.Stage; importer javafx.util.Callback;
```

```
la classe publique ListViewSample étend l'application {

    ListView<String> list = new ListView<String>(); ObservableList<String>
    data = FXCollections.observableArrayList( "chocolate", "salmon", "gold", "corail", "darkorchid",
        "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue", "bleu violet", "marron");
```

```
@Override
public void start(Stage stage) {
    Boîte VBox = nouvelle VBox();
    Scene scene = new Scene(box, 200, 200);
    stage.setScene(scene);
    stage.setTitle("ListViewSample");
```

```

box.getChildren().addAll(liste);
VBox.setVgrow(list, Priority.ALWAYS);

list.setItems (données);

list.setCellFactory(new Callback<ListView<String>, ListCell<String>>() {

    @Passer outre
    public ListCell<String> call(ListView<String> list) {
        return new ColorRectCell();
    }
});

spectacle();
}

la classe statique ColorRectCell étend ListCell<String> {
    @Passer outre
    public void updateItem (élément de chaîne, booléen vide) {
        super.updateItem(élément, vide);
        Rectangle rect = nouveau Rectangle(100, 20);
        si (élément != null) {
            rect.setFill(Color.web(item));
            setGraphic(rect);
        }
    }
}

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

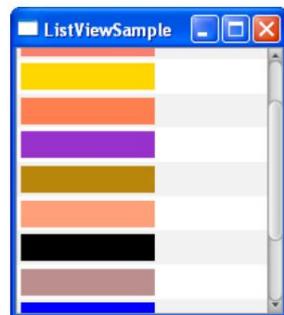
```

La fabrique de cellules produit des objets ListCell. Chaque cellule est associée à un seul élément de données et affiche une seule "ligne" de la vue de liste. Le contenu représenté par la cellule via la méthode setGraphic peut inclure d'autres contrôles, du texte, des formes ou des images.

Dans cette application, la cellule de liste affiche des rectangles.

La compilation et l'exécution de l'application produisent la fenêtre illustrée à la [Figure 11–5](#).

**Figure 11–5 Liste des motifs de couleur**



Vous pouvez faire défiler la liste, sélectionner et désélectionner n'importe lequel de ses éléments. Vous pouvez également étendre cette application pour remplir l'étiquette de texte avec le motif de couleur comme indiqué dans la section suivante.

## Traitement de la sélection d'éléments de liste

Modifiez le code d'application comme indiqué dans l'[exemple 11–5](#) pour activer le traitement de l'événement lorsqu'un élément de liste particulier est sélectionné.

### Exemple 11–5 Traitement d'événements pour un élément de liste

```
importer javafx.application.Application; importer
javafx.beans.value.ChangeListener<String>; importez
javafx.beans.value.ObservableValue<String>; importer
javafx.collections.FXCollections<String>; importer
javafx.collections.ObservableList<String>; importer javafx.scene.Scene;
importer javafx.scene.control.Label; importer
javafx.scene.control.ListCell; importer
javafx.scene.control.ListView; importer
javafx.scene.layout.Priority; importer javafx.scene.layout.VBox;
importer javafx.scene.paint.Color; importer
javafx.scene.shape.Rectangle; importer javafx.scene.text.Font;
importer javafx.stage.Stage; importer javafx.util.Callback;
```

la classe publique ListViewSample étend l'application {

```
ListView<String> list = new ListView<String>(); ObservableList<String>
data = FXCollections.observableArrayList("chocolate", "salmon", "gold", "corail", "darkorchid",
"darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue", "bleu violet", "marron");
```

**Étiquette finale étiquette = nouvelle étiquette();**

```
@Passer outre
public void start(Stage stage) {
    Boîte VBox = nouvelle VBox();
    Scene scene = new Scene(box, 200, 200);
    stage.setScene(scene);
    stage.setTitle("ListViewSample");
    box.getChildren().addAll(liste, étiquette);
    VBox.setVgrow(list, Priority.ALWAYS);

    label.setLayoutX(10);
    label.setLayoutY(115);
    label.setFont(Font.font("Verdana", 20));

    list.setItems (données);

    list.setCellFactory(new Callback<ListView<String>, ListCell<String>>()
    { @Override

        public ListCell<String> call(ListView<String> list) {
            return new ColorRectCell();
        }
    });
};

list.getSelectionModel().selectedItemProperty().addListener(
    new ChangeListener<String>() {
        public void a changé (ObservableValue<? étend la chaîne> ov,
```

```

        Chaîne ancienne_val, Chaîne nouvelle_val) {
            label.setText(new_val);
            label.setTextFill(Color.web(new_val));
        }
    });
spectacle();
}

la classe statique ColorRectCell étend ListCell<String> {
    @Passer autre
    public void updateItem (élément de chaîne, boolean vide) { super.updateItem
        (élément, vide); Rectangle rect = nouveau Rectangle(100, 20); si
        (élément != null) {

            rect.setFill(Color.web(item)); setGraphique(rect);

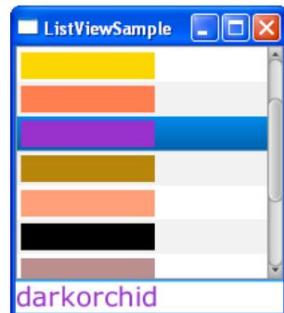
        }
    }
}

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

```

La méthode addListener appelée pour le selectedItemProperty crée un nouvel objet ChangeListener<String> pour gérer les modifications de l'élément sélectionné. Si, par exemple, l'élément orchidée noire est sélectionné, l'étiquette reçoit la légende « darkorchid » et est remplie avec la couleur correspondante. La sortie de l'application modifiée est illustrée à la [Figure 11–6](#).

Figure 11–6 Sélection d'un motif de couleur d'orchidée foncée



#### Documentation relative à l'API

- ÿ ListView
- ÿ CelluleListe
- ÿ ComboBoxListCell

Traitement de la sélection d'éléments de liste

---

12

## 12Vue du tableau

Dans ce chapitre, vous apprendrez à effectuer des opérations de base avec des tables dans les applications JavaFX, telles que l'ajout d'une table, le remplissage de la table avec des données et la modification de la table.

Plusieurs classes de l'API JavaFX SDK sont conçues pour représenter les données sous forme de tableau. Les classes les plus importantes pour créer des tables dans les applications JavaFX sont TableView, TableColumn et TableCell. Vous pouvez remplir une table en implémentant le modèle de données et en appliquant une fabrique de cellules.

Les classes de table fournissent des fonctionnalités intégrées pour trier les données dans les colonnes et redimensionner les colonnes si nécessaire.

La Figure 12-1 montre un tableau typique représentant les informations de contact d'un carnet d'adresses.

**Figure 12–1 Exemple de tableau**



## Création d'un tableau

Le fragment de code de l' [exemple 12–1](#) crée une table vide à trois colonnes et l'ajoute à la scène de l'application.

### Exemple 12–1 Ajout d'un tableau

```
importer javafx.application.Application; importer
javafx.geometry.Insets; importer javafx.scene.Group;
importer javafx.scene.Scene; importer
javafx.scene.control.Label; importer
javafx.scene.control.TableColumn; importer
javafx.scene.control.TableView; importer
javafx.scene.layout.VBox; importer javafx.scene.text.Font;
importer javafx.stage.Stage;
```

```
la classe publique TableViewSample étend l'application {

    table TableView privée = new TableView(); public static void
main(String[] args) { launch(args);

}

    @Passer outre
public void start(Stage stage) { Scene scene =
    new Scene(new Group()); stage.setTitle("Exemple de
vue de tableau"); stage.setWidth(300);
    stage.setHeight(500);

    label final label = new Label("Carnet d'adresses");
    label.setFont(nouvelle police("Arial", 20));

    table.setEditable(true);

    TableColumn firstNameCol = new TableColumn("First Name");
    TableColumn lastNameCol = new TableColumn("Last Name");
    TableColumn emailCol = new TableColumn("Email");

    table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

    VBox finale vbox = nouvelle VBox();
    vbox.setSpacing(5);
    vbox.setPadding(nouveaux inserts(10, 0, 0, 10));
    vbox.getChildren().addAll(étiquette, table);

    ((Groupe) scene.getRoot()).getChildren().addAll(vbox);

    stage.setScene(scene);
    spectacle();
}
}
```

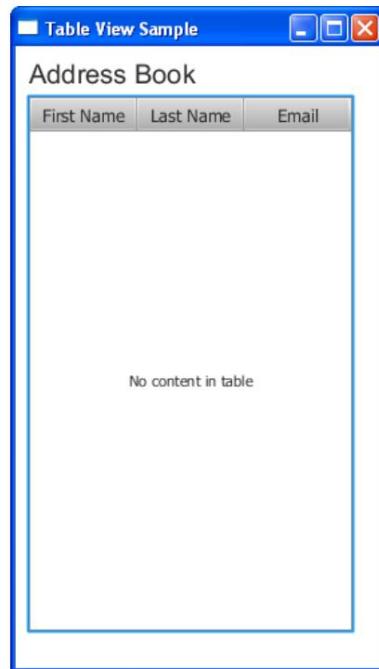
Le contrôle table est créé en instantiant la classe TableView. Dans l' [exemple 12–1](#), il est ajouté au conteneur de mise en page VBox, cependant, vous pouvez l'ajouter directement à la scène de l'application.

L'[exemple 12–1](#) définit trois colonnes pour stocker les informations suivantes dans un carnet d'adresses : le prénom et le nom d'un contact et une adresse e-mail. Les colonnes sont créées à l'aide de la classe TableColumn.

La méthode getColumns de la classe TableView ajoute les colonnes précédemment créées à la table. Dans vos applications, vous pouvez utiliser cette méthode pour ajouter et supprimer dynamiquement des colonnes.

La compilation et l'exécution de cette application produisent la sortie illustrée à la [Figure 12–2](#).

**Figure 12–2 Tableau sans données**



Vous pouvez gérer la visibilité des colonnes en appelant la méthode setVisible. Par exemple, si la logique de votre application nécessite de masquer les adresses e-mail des utilisateurs, vous pouvez implémenter cette tâche comme suit: emailCol.setVisible(false).

Lorsque la structure de vos données nécessite une représentation plus compliquée, vous pouvez créer des colonnes imbriquées.

Par exemple, supposons que les contacts du carnet d'adresses disposent de deux comptes de messagerie. Ensuite, vous avez besoin de deux colonnes pour afficher les adresses e-mail principale et secondaire. Créez deux sous-colonnes et appelez la méthode getColumns sur emailCol, comme illustré dans l' [exemple 12–2](#).

### Exemple 12–2 Création de colonnes imbriquées

```
TableColumn firstEmailCol = new TableColumn("Primary");
TableColumn secondEmailCol = new TableColumn("Secondary");

emailCol.getColumns().addAll(firstEmailCol, secondEmailCol);
```

Une fois que vous avez ajouté ces lignes à l' [exemple 12–1](#), et que vous avez compilé et exécuté le code de l'application, le tableau apparaît comme illustré à la [figure 12–3](#).

**Figure 12–3 Tableau avec colonnes imbriquées**



Bien que la table soit ajoutée à l'application, la légende standard "Pas de contenu dans la table" s'affiche, car aucune donnée n'est définie. Au lieu d'afficher cette légende, vous pouvez utiliser la méthode `setPlaceholder` pour spécifier un objet `Node` à afficher dans un tableau vide.

## Définition du modèle de données

Lorsque vous créez une table dans une application JavaFX, il est recommandé d'implémenter une classe qui définit le modèle de données et fournit des méthodes et des champs pour continuer à travailler avec la table. L' [exemple 12–3](#) crée la classe `Person` pour définir des données dans un carnet d'adresses.

### Exemple 12–3 Crédit de la classe Person

```
public static class Personne {
    private final SimpleStringProperty firstName;
    private final SimpleStringProperty lastName;
    private final SimpleStringProperty email;

    public Personne(String fName, String lName, String email) {
        this.firstName = new SimpleStringProperty(fName);
        this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);
    }

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

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

```

    }

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

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

    public String getEmail() { return
        email.get();
    }

    public void setEmail(String fName) {
        email.set(fName);
    }
}

```

Les propriétés de chaîne firstName, lastName et email sont créées pour permettre le référencement d'un élément de données particulier.

De plus, les méthodes get et set sont fournies pour chaque élément de données. Ainsi, par exemple, la méthode getFirstName renvoie la valeur de la propriété firstName et la méthode setFirstName spécifie une valeur pour cette propriété.

Lorsque le modèle de données est décrit dans la classe Person, vous pouvez créer un tableau ObservableList et définir autant de lignes de données que vous souhaitez afficher dans votre tableau. Le fragment de code de l' [exemple 12–4](#) implémente cette tâche.

#### Exemple 12–4 Définition de données de table dans une liste observable

```

final ObservableList<Person> data = FXCollections.observableArrayList( new Person("Jacob", "Smith",
    "jacob.smith@example.com"), new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
    new Person("Ethan", "Williams", "ethan.williams@example.com"), new Person("Emma", "Jones",
    "emma.jones@example.com"), new Person("Michael", "Brown", "michael.brown@example.com")
);


```

L'étape suivante consiste à associer les données aux colonnes du tableau. Vous pouvez le faire via les propriétés définies pour chaque élément de données, comme illustré dans l' [exemple 12–5](#).

#### Exemple 12–5 Définition des propriétés de données sur des colonnes

```

firstNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("firstName")
);
lastNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("lastName")
);
emailCol.setCellValueFactory(
    new PropertyValueFactory<Person, String>("email")
);

```

La méthode setCellValueFactory spécifie une fabrique de cellules pour chaque colonne. Les fabriques de cellules sont implémentées à l'aide de la classe PropertyValueFactory, qui utilise les propriétés firstName, lastName et email des colonnes de la table comme références aux méthodes correspondantes de la classe Person.

Lorsque le modèle de données est défini et que les données sont ajoutées et associées aux colonnes, vous pouvez ajouter les données à la table à l'aide de la méthode `setItems` de la classe `TableView` : `table.setItems(data)`.

Étant donné que l'objet `ObservableList` permet le suivi de toute modification de ses éléments, le contenu `TableView` est automatiquement mis à jour chaque fois que les données changent.

Examinez le code d'application illustré dans l'[exemple 12–6](#).

**Exemple 12–6** Création d'une table et ajout de données à celle-ci

```
importer javafx.application.Application; importer
javafx.beans.property.SimpleStringProperty; importer
javafx.collections.FXCollections; importer javafx.collections.ObservableList;
importer javafx.geometry.Insets; importer javafx.scene.Group; importer
javafx.scene.Scene; importer javafx.scene.control.Label; importer
javafx.scene.control.TableColumn; importer javafx.scene.control.TableView;
importer javafx.scene.control.TextField; importer
javafx.scene.control.cell.PropertyValueFactory; importer
javafx.scene.layout.VBox; importer javafx.scene.text.Font; importer
javafx.stage.Stage;
```

la classe publique `TableViewSample` étend l'application {

```
table TableView privée<Personne> = new TableView<Personne>(); données finales
privées ObservableList<Person> = FXCollections.observableArrayList(
    new Person("Jacob", "Smith", "jacob.smith@example.com"), new Person("Isabella",
    "Johnson", "isabella.johnson@example.com"), new Person("Ethan", "Williams",
    "ethan.williams@example.com"), new Person("Emma", "Jones", "emma.jones@example.com"),
    new Person("Michael", "Brown", "michael.brown@example.com")
);
public static void main(String[] args) { launch(args);
}
@Passer outre
public void start(Stage stage) { Scene scene =
    new Scene(new Group()); stage.setTitle("Exemple de
    vue de tableau"); stage.setWidth(450);
    stage.setHeight(500);

    label final label = new Label("Carnet d'adresses");
    label.setFont(nouvelle police("Arial", 20));

    table.setEditable(true);

    TableColumn firstNameCol = new TableColumn("First Name");
    firstNameCol.setMinWidth(100); firstNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("firstName"));

    TableColumn lastNameCol = new TableColumn("Last Name");
}
```

```
lastNameCol.setMinWidth(100);
lastNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("lastName"));

TableColumn emailCol = new TableColumn("Email");
emailCol.setMinWidth(200);
emailCol.setCellValueFactory(
    new PropertyValueFactory<Person, String>("email"));

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

VBox finale vbox = nouvelle VBox();
vbox.setSpacing(5);
vbox.setPadding(nouveaux inserts(10, 0, 0, 10));
vbox.getChildren().addAll(étiquette, table);

((Groupe) scene.getRoot()).getChildren().addAll(vbox);

stage.setScene(scene);
spectacle();
}

public static class Personne {

    private final SimpleStringProperty firstName; private final
    SimpleStringProperty lastName; e-mail final privé
    SimpleStringProperty;

    personne privée (chaîne fName, chaîne lName, chaîne email) { this.firstName = new
        SimpleStringProperty (fName); this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);

    }

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

    public void setFirstName(String fName) { firstName.set(fName);

    }

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

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

    public String getEmail() { return
        email.get();
    }

    public void setEmail(String fName) { email.set(fName);

    }
}
```

Ajout de nouvelles lignes

Lorsque vous compilez et exécutez ce code d'application, le tableau illustré à la [Figure 12–4](#) s'affiche.

**Figure 12–4 Tableau rempli de données**

The screenshot shows a JavaFX application window titled "Table View Sample". Inside, there is a title bar "Address Book". Below it is a table with three columns: "First Name", "Last Name", and "Email". The table contains the following data:

First Name	Last Name	Email
Jacob	Smith	jacob.smith@example.com
Isabella	Johnson	isabella.johnson@example.com
Ethan	Williams	ethan.williams@example.com
Emma	Jones	emma.jones@example.com
Michael	Brown	michael.brown@example.com

## Ajout de nouvelles lignes

Le tableau de la [Figure 12–4](#) contient cinq lignes de données, qui ne peuvent pas être modifiées jusqu'à présent.

Vous pouvez utiliser des champs de texte pour saisir de nouvelles valeurs dans les colonnes Prénom, Nom et E-mail. Le contrôle [Champ de texte](#) permet à votre application de recevoir une entrée de texte d'un utilisateur.

[L'exemple 12–7](#) crée trois champs de texte, définit le texte d'invite pour chaque champ et crée le bouton Ajouter.

### Exemple 12–7 Utilisation de champs de texte pour entrer de nouveaux éléments dans le tableau

```
final TextField addFirstName = new TextField();
addFirstName.setPromptText("Prénom");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth()); final TextField
addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Nom"); final TextField addEmail = new TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth()); addEmail.setPromptText("Email");
```

```
Bouton final addButton = new Button("Ajouter");
addButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
        data.add(nouvelle
            Personne( addFirstName.getText(),
            addLastName.getText(),
            addEmail.getText()
        ));
    }
});
```

```

        addFirstName.clear();
        addLastName.clear();
        addEmail.clear();
    }
});

```

Lorsqu'un utilisateur clique sur le bouton Ajouter, les valeurs saisies dans les champs de texte sont incluses dans un constructeur Person et ajoutées à la liste des données observables. Ainsi, la nouvelle entrée avec les informations de contact apparaît dans le tableau.

Examinez le code d'application illustré dans l' [exemple 12–8](#).

#### Exemple 12–8 Tableau avec les champs de texte pour saisir de nouveaux éléments

```

importer javafx.application.Application; importer
javafx.beans.property.SimpleStringProperty; importer
javafx.collections.FXCollections; importer javafx.collections.ObservableList;
import javafx.event.ActionEvent; import javafx.event.EventHandler;
importer javafx.geometry.Insets; importer javafx.scene.Group; importer
javafx.scene.Scene; importer javafx.scene.control.Button; importer
javafx.scene.control.Label; importer javafx.scene.control.TableColumn;
importer javafx.scene.control.TableView; importer
javafx.scene.control.TextField; importer
javafx.scene.control.cell.PropertyValueFactory; importer
javafx.scene.layout.HBox; importer javafx.scene.layout.VBox; importer
javafx.scene.text.Font; importer javafx.stage.Stage;

```

la classe publique FileChooserSample étend l'application {

```

table TableView privée<Personne> = new TableView<Personne>(); données finales
privées ObservableList<Person> =
    FXCollections.observableArrayList( new
        Person("Jacob", "Smith", "jacob.smith@example.com"), new Person("Isabella",
        "Johnson", "isabella.johnson@example.com"), new Person ("Ethan", "Williams",
        "ethan.williams@example.com"), nouvelle Personne("Emma", "Jones", "emma.jones@example.com"),
        nouvelle Personne("Michael", " Brown", "michael.brown@example.com"));
}

HBox final hb = new HBox();

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

@Passer autre
public void start(Stage stage) { Scene scene =
    new Scene(new Group()); stage.setTitle("Exemple de
    vue de tableau"); stage.setWidth(450);
    stage.setHeight(550);

    label final label = new Label("Carnet d'adresses");
    label.setFont(nouvelle police("Arial", 20));

    table.setEditable(true);
}

```

Ajout de nouvelles lignes

```

TableColumn firstNameCol = new TableColumn("First Name");
firstNameCol.setMinWidth(100);
firstNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("firstName"));

TableColumn lastNameCol = new TableColumn("Last Name");
lastNameCol.setMinWidth(100);
lastNameCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("lastName"));

TableColumn emailCol = new TableColumn("Email");
emailCol.setMinWidth(200); emailCol.setCellValueFactory(
    new PropertyValueFactory<Person, String>("email"));

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

final TextField addFirstName = new TextField();
addFirstName.setPromptText("Prénom");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth()); final TextField
addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Nom"); final TextField addEmail = new
TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth());
addEmail.setPromptText("Email");

Bouton final addButton = new Button("Ajouter");
addButton.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) { data.add(new
        Person(
            addFirstName.getText(),
            addLastName.getText(),
            addEmail.getText());
        addFirstName.clear();
        addLastName.clear();
        addEmail.clear();
    }
});
});

hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton); hb.setSpacing(3);

VBox finale vbox = nouvelle VBox();
vbox.setSpacing(5);
vbox.setPadding(nouveaux inserts(10, 0, 0, 10));
vbox.getChildren().addAll(label, table, hb);

((Groupe) scene.getRoot()).getChildren().addAll(vbox);

stage.setScene(scene);
spectacle();
}

public static class Personne {

    private final SimpleStringProperty firstName;

```

```
private final SimpleStringProperty lastName; e-mail final privé
SimpleStringPropertylastName;

personne privée (chaîne fName, chaîne lName, chaîne email) { this.firstName = new
    SimpleStringProperty (fName); this.lastName = new SimpleStringProperty(lName);
    this.email = new SimpleStringProperty(email);

}

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

public void setFirstName(String fName) { firstName.set(fName);

}

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

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

public String getEmail() { return
    email.get();
}

public void setEmail(String fName) { email.set(fName);

}

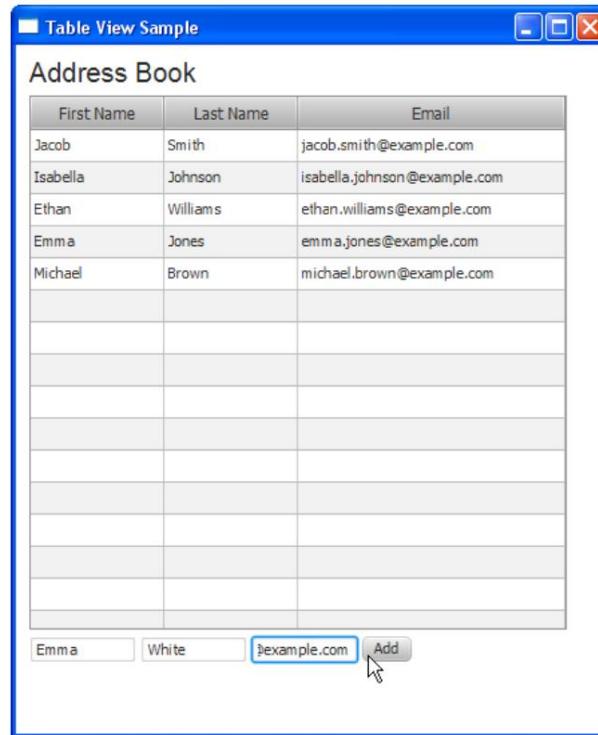
}
}
```

Cette application ne fournit aucun filtre pour vérifier si, par exemple, une adresse e-mail a été saisie dans un format incorrect. Vous pouvez fournir une telle fonctionnalité lorsque vous développez votre propre application.

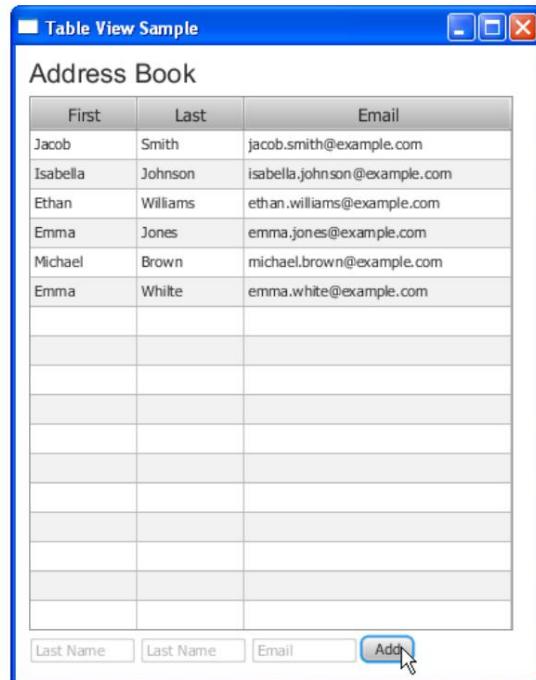
L'implémentation actuelle ne vérifie pas non plus si les valeurs vides sont saisies. Si aucune valeur n'est fournie, cliquer sur le bouton Ajouter insère une ligne vide dans le tableau.

[La Figure 12–5](#) montre comment un utilisateur ajoute une nouvelle ligne de données.

Ajout de nouvelles lignes

**Figure 12–5 Ajout d'informations de contact au carnet d'adresses**

La [Figure 12–6](#) montre le tableau après avoir cliqué sur le bouton Ajouter. Les coordonnées d'Emma White apparaissent désormais dans le tableau.

**Figure 12–6 Entrée nouvellement ajoutée**

### Trier les données en colonnes

La classe `TableView` fournit des fonctionnalités intégrées pour trier les données dans des colonnes. Les utilisateurs peuvent modifier l'ordre des données en cliquant sur les en-têtes de colonne. Le premier clic active l'ordre de tri croissant, le deuxième clic active l'ordre de tri décroissant et le troisième clic désactive le tri. Par défaut, aucun tri n'est appliqué.

Les utilisateurs peuvent trier plusieurs colonnes dans une table et spécifier la priorité de chaque colonne dans l'opération de tri. Pour trier plusieurs colonnes, l'utilisateur appuie sur la touche Maj tout en cliquant sur l'en-tête de chaque colonne à trier.

Dans la Figure 12-7, un ordre de tri croissant est appliqué aux prénoms, tandis que les noms de famille sont triés par ordre décroissant. Notez que la première colonne a priorité sur la deuxième colonne.

### **Figure 12–7 Tri de plusieurs colonnes**

En tant que développeur d'application, vous pouvez définir des préférences de tri pour chaque colonne de votre application en appliquant la méthode `setSortType`. Vous pouvez spécifier à la fois le type croissant et décroissant. Par exemple, utilisez la ligne de code suivante pour définir le type de tri décroissant pour la colonne `emailCol`:

```
emailCol.setSortType(TableColumn.SortType.DESCENDING);
```

Vous pouvez également spécifier les colonnes à trier en ajoutant et en supprimant TableColumn instances de la liste observable TableView.sortOrder. L'ordre des colonnes dans cette liste représente la priorité de tri (par exemple, l'élément zéro a une priorité plus élevée que le premier élément).

Pour interdire le tri des données, appelez la méthode `setSortable(false)` sur la colonne.

## Modification des données dans le tableau

La classe `TableView` rend non seulement les données tabulaires, mais elle fournit également des fonctionnalités pour les modifier. Utilisez la méthode `setEditable` pour activer la modification du contenu du tableau.

Utilisez la méthode `setCellFactory` pour réimplémenter la cellule du tableau en tant que champ de texte à l'aide de la classe `TextFieldTableCell`. La méthode `setOnEditCommit` traite la modification et affecte la valeur mise à jour à la cellule de tableau correspondante.

L'[exemple 12–9](#) montre comment appliquer ces méthodes pour traiter la modification de cellule dans les colonnes Prénom, Nom et E-mail.

#### **Exemple 12–9 Implémentation de l'édition de cellule**

```
firstNameCol.setCellFactory(TextFieldTableCell.forTableColumn()); firstNameCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne) t.getTableView().getItems().get(
                t.getTablePosition().getRow()
            )).setFirstName(t.getNewValue());
        }
    }
);

lastNameCol.setCellFactory(TextFieldTableCell.forTableColumn()); lastNameCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne) t.getTableView().getItems().get(
                t.getTablePosition().getRow()
            )).setLastName(t.getNewValue());
        }
    }
);

emailCol.setCellFactory(TextFieldTableCell.forTableColumn()); emailCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne) t.getTableView().getItems().get(
                t.getTablePosition().getRow()
            )).setEmail(t.getNewValue());
        }
    }
);
```

Le code complet de l'application illustrée dans l' [exemple 12–10](#).

#### **Exemple 12–10 TableViewSample avec modification de cellule activée**

```
importer javafx.application.Application;
importer javafx.beans.property.SimpleStringProperty;
importer javafx.collections.FXCollections;
importer javafx.collections.ObservableList;
importer javafx.event.ActionEvent;
importer javafx.event.EventHandler;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Button;
importer javafx.scene.control.Label;
importer javafx.scene.control.TableColumn;
importer javafx.scene.control.TableColumn.CellEditEvent;
importer javafx.scene.control.TableView;
```

```
importer javafx.scene.control.TextField; importer
javafx.scene.control.cell.PropertyValueFactory; importer
javafx.scene.control.cell.TextFieldTableCell; importer javafx.scene.layout.HBox;
importer javafx.scene.layout.VBox; importer javafx.scene.text.Font; importer
javafx.stage.Stage;
```

la classe publique TableViewSample étend l'application {

```
table TableView privée<Personne> = new TableView<Personne>(); données finales
privées ObservableList<Person> =
    FXCollections.observableArrayList( new
        Person("Jacob", "Smith", "jacob.smith@example.com"), new Person("Isabella",
        "Johnson", "isabella.johnson@example.com"), new Person ("Ethan", "Williams",
        "ethan.williams@example.com"), nouvelle Personne("Emma", "Jones", "emma.jones@example.com"),
        nouvelle Personne("Michael", " Brown", "michael.brown@example.com") );
```

```
HBox final hb = new HBox();
```

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

```
}
```

```
@Passer autre
public void start(Stage stage) { Scene scene =
    new Scene(new Group()); stage.setTitle("Exemple de
    vue de tableau"); stage.setWidth(450);
    stage.setHeight(550);
```

```
label final label = new Label("Carnet d'adresses");
label.setFont(nouvelle police("Arial", 20));
```

```
table.setEditable(true);
```

```
TableColumn firstNameCol = new TableColumn("First Name");
firstNameCol.setMinWidth(100);
firstNameCol.setCellValueFactory(
    new PropertyValueFactory<Person, String>("firstName"));
firstNameCol.setCellFactory(TextFieldTableCell.forTableColumn()); firstNameCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer autre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne) t.getTableView().getItems().get(
                t.getTablePosition().getRow() )).setFirstName(t.getNewValue());
        }
    });
};
```

```
TableColumn lastNameCol = new TableColumn("Last Name");
lastNameCol.setMinWidth(100); lastNameCol.setCellValueFactory(
    new PropertyValueFactory<Person, String>("lastName"));
lastNameCol.setCellFactory (TextFieldTableCell.forTableColumn ()); lastNameCol.setOnEditCommit
(
    new EventHandler<CellEditEvent<Person, String>>() {
```

```

    @Passer outre
    public void handle(CellEditEvent<Person, String> t) {
        ((Personne)

            t.getTableView().getItems().get( t.getTablePosition().getRow() ).setLastName(t.getNewValue());
        }
    );
}

TableColumn emailCol = new TableColumn("Email");
emailCol.setMinWidth(200);
emailCol.setCellValueFactory( new
    PropertyValueFactory<Person, String>("email"));
emailCol.setCellFactory (TextFieldTableCell.forTableColumn () ); emailCol.setOnEditCommit
(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne)

                t.getTableView().getItems().get( t.getTablePosition().getRow() ).setEmail(t.getNewValue());
            }
        }
    );
);

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

final TextField addFirstName = new TextField();
addFirstName.setPromptText("Prénom");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth()); final TextField
addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Nom"); final TextField addEmail = new
TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth());
addEmail.setPromptText("Email");

Bouton final addButton = new Button("Ajouter");
addButton.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) { data.add(new
        Person(
            addFirstName.getText(),
            addLastName.getText(),
            addEmail.getText());
        addFirstName.clear();
        addLastName.clear();
        addEmail.clear();
    )
});}

hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton); hb.setSpacing(3);

VBox finale vbox = nouvelle VBox();
vbox.setSpacing(5);
vbox.setPadding(nouveaux inserts(10, 0, 0, 10));
vbox.getChildren().addAll(label, table, hb);

```

```
((Groupe) scene.getRoot()).getChildren().addAll(vbox);

stage.setScene(scene);
spectacle();
}

public static class Personne {

    private final SimpleStringProperty firstName; private final
    SimpleStringProperty lastName; e-mail final privé
    SimpleStringProperty;

    personne privée (chaîne fName, chaîne lName, chaîne email) { this.firstName = new
        SimpleStringProperty (fName); this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);

    }

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

    public void setFirstName(String fName) { firstName.set(fName);

    }

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

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

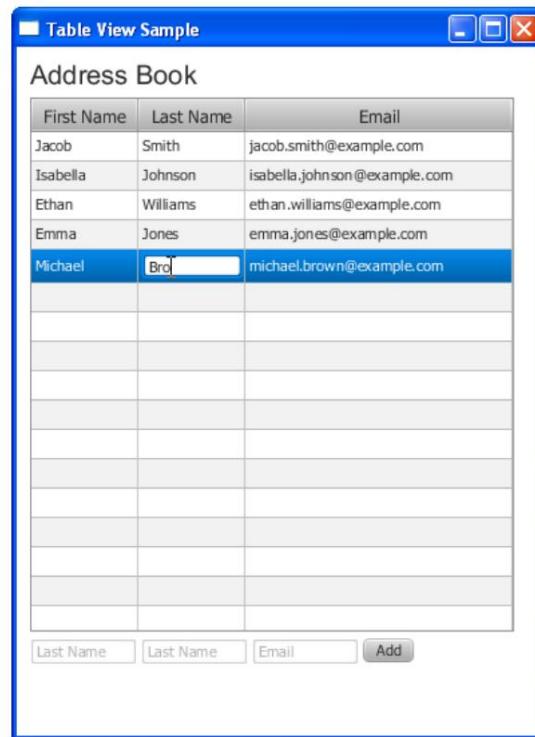
    public String getEmail() { return
        email.get();
    }

    public void setEmail(String fName) { email.set(fName);

    }
}
}
```

Dans la [Figure 12–8](#), l'utilisateur modifie le nom de famille de Michael Brown. Pour modifier une cellule du tableau, l'utilisateur saisit la nouvelle valeur dans la cellule, puis appuie sur la touche Entrée. La cellule n'est pas modifiée tant que la touche Entrée n'est pas enfoncée. Ce comportement est déterminé par l'implémentation de la classe `TextField`.

Figure 12–8 Modification d'une cellule de tableau



Notez que l'implémentation par défaut du contrôle `TextField` nécessite que les utilisateurs appuient sur la touche Entrée pour valider la modification. Vous pouvez redéfinir le comportement `TextField` pour valider la modification sur le changement de focus, ce qui est une expérience utilisateur attendue. Essayez le code modifié pour implémenter un tel comportement alternatif.

#### Exemple 12–11 Solution alternative d'édition de cellule

```
importer javafx.application.Application; importer
javafx.beans.property.SimpleStringProperty; importer
javafx.beans.value.ChangeListener; importez
javafx.beans.value.ObservableValue; importer
javafx.collections.FXCollections; importer
javafx.collections.ObservableList; import javafx.event.ActionEvent;
import javafx.event.EventHandler; importer javafx.geometry.Insets;
importer javafx.scene.Group; importer javafx.scene.Scene;
importer javafx.scene.control.Button; importer
javafx.scene.control.Label; importer javafx.scene.control.TableCell;
importer javafx.scene.control.TableColumn; importer
javafx.scene.control.TableColumn.CellEditEvent; importer
javafx.scene.control.TableView; importer
javafx.scene.control.TextField; importer
javafx.scene.control.cell.PropertyValueFactory; importer
javafx.scene.layout.HBox; importer javafx.scene.layout.VBox;
```

```

importer javafx.scene.text.Font;
importer
javafx.stage.Stage;
importer
javafx.util.Callback;

la classe publique TableViewSample étend l'application {

    table TableView privée<Personne> = new TableView<Personne>(); données finales
    privées ObservableList<Person> =
        FXCollections.observableArrayList( new
            Person("Jacob", "Smith", "jacob.smith@example.com"), new Person("Isabella",
            "Johnson", "isabella.johnson@example.com"), new Person ("Ethan", "Williams",
            "ethan.williams@example.com"), nouvelle Personne("Emma", "Jones", "emma.jones@example.com"),
            nouvelle Personne("Michael", " Brown", "michael.brown@example.com"));

    HBox final hb = new HBox();

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

    }

    @Passer outre
    public void start(Stage stage) { Scene scene =
        new Scene(new Group()); stage.setTitle("Exemple de
        vue de tableau"); stage.setWidth(450);
        stage.setHeight(550);

        label final label = new Label("Carnet d'adresses");
        label.setFont(nouvelle police("Arial", 20));

        table.setEditable(true);
        Callback<TableColumn, TableCell> cellFactory = new
            Callback<TableColumn, TableCell>() {
                public TableCell call(TableColumn p) { return new
                    EditingCell();
                }
            };
        }

        TableColumn firstNameCol = new TableColumn("First Name");
        firstNameCol.setMinWidth(100); firstNameCol.setCellValueFactory(
            new PropertyValueFactory<Person, String>"firstName"));
        firstNameCol.setCellFactory(cellFactory); firstNameCol.setOnEditCommit(
            new EventHandler<CellEditEvent<Person, String>>() {
                @Passer outre
                public void handle(CellEditEvent<Person, String> t) {
                    ((Personne)
                        t.getTableView().getItems().get( t.getTablePosition().getRow() )).setFirstName(t.getNewValue());
                }
            });
        );

        TableColumn lastNameCol = new TableColumn("Last Name");
        lastNameCol.setMinWidth(100); lastNameCol.setCellValueFactory(
            new PropertyValueFactory<Person, String>"lastName"));
        lastNameCol.setCellFactory(cellFactory);
    }
}

```

```

lastNameCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne)

                t.getTableView().getItems().get( t.getTablePosition().getRow() ).setLastName(t.getNewValue());
            }
        }
);

TableColumn emailCol = new TableColumn("Email");
emailCol.setMinWidth(200); emailCol.setCellValueFactory( new
PropertyValueFactory<Person, String>("email"));

emailCol.setCellFactory(cellFactory);
emailCol.setOnEditCommit(
    new EventHandler<CellEditEvent<Person, String>>() {
        @Passer outre
        public void handle(CellEditEvent<Person, String> t) {
            ((Personne)

                t.getTableView().getItems().get( t.getTablePosition().getRow() ).setEmail(t.getNewValue());
            }
        }
);

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

final TextField addFirstName = new TextField();
addFirstName.setPromptText("Prénom");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth()); final TextField
addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Nom"); final TextField addEmail = new TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth()); addEmail.setPromptText("Email");



Bouton final addButton = new Button("Ajouter");
addButton.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) { data.add(new
Person(
        addFirstName.getText(),
        addLastName.getText(),
        addEmail.getText());
    addFirstName.clear();
    addLastName.clear();
    addEmail.clear();
)
});
});

hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton); hb.setSpacing(3);

VBox finale vbox = nouvelle VBox();
vbox.setSpacing(5);
vbox.setPadding(nouveaux inserts(10, 0, 0, 10));

```

```
vbox.getChildren().addAll(label, table, hb);

((Groupe) scene.getRoot()).getChildren().addAll(vbox);

stage.setScene(scene);
spectacle();
}

public static class Personne {

    private final SimpleStringProperty firstName; private final
    SimpleStringProperty lastName; e-mail final privé
    SimpleStringProperty;

    personne privée (chaîne fName, chaîne lName, chaîne email) { this.firstName = new
        SimpleStringProperty (fName); this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);

    }

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

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

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

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

    public String getEmail() { return
        email.get();
    }

    public void setEmail(String fName) { email.set(fName);

    }
}

class EditingCell étend TableCell<Person, String> {

    champ de texte privé TextField;

    public EditingCell() { }

    @Passer outre
    public void startEdit() { if (!isEmpty())
        { super.startEdit();
            createTextField();
            setText(null);
            setGraphic(textField);
            textField.selectAll();
        }
    }
}
```

```

    }

    @Passer autre
    public void cancelEdit()
    {
        super.cancelEdit();

        setText((String) getItem()); setGraphic(null);

    }

    @Passer autre
    public void updateItem (élément de chaîne, booléen vide) { super.updateItem
        (élément, vide);

        si (vide) {
            setText(null);
            setGraphic(null); } autre
        {
            if (isEditing()) { if (textField !
                = null) {
                    textField.setText(getString());
                }
                setText(null);
                setGraphic(textField); } autre {

                    setText(getString());
                    setGraphic(null);
                }
        }
    }

privé void createTextField() { textField = new
    TextField(getString()); textField.setMinWidth(this.getWidth()
    - this.getGraphicTextGap()* 2); textField.focusedProperty().addListener(new ChangeListener<Boolean>(){

{
    @Passer autre
    public void modifié (ObservableValue<? étend Boolean> arg0,
        Boolean arg1, Boolean arg2) { if (!arg2)
            { commitEdit(textField.getText());

            }
        }
    });

}

chaîne privée getString() {
    retourner getItem() == null ? ""
        : getItem().toString();
}

```

Notez que cette approche pourrait devenir redondante dans les versions futures à mesure que l'implémentation de `TextFieldTableCell` évolue pour offrir une meilleure expérience utilisateur.

## Ajout de cartes de données à la table

JavaFX SDK 2.2, vous pouvez ajouter les données de carte à la table. Utilisez la classe MapValueFactory comme illustré dans l' [exemple 12-12](#) pour afficher la carte des ID d'étudiants dans le tableau.

### Exemple 12-12 Ajout de données cartographiques au tableau

```
importer java.util.HashMap;
importer java.util.Map;
importer javafx.application.Application;
importer javafx.collections.FXCollections;
importer javafx.collections.ObservableList;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Label;
importer javafx.scene.control.TableCell;
importer javafx.scene.control.TableColumn;
importer javafx.scene.control.TableView;
importer javafx.scene.control.cell.MapValueFactory;
importer javafx.scene.control.cell.TextFieldTableCell;
importer javafx.scene.layout.VBox;
importer javafx.scene.text.Font;
importer javafx.stage.Stage;
importer javafx.util.Callback;
importer javafx.util.StringConverter;
```

```
la classe publique TableViewSample étend l'application {
```

```
    public static final String Column1MapKey = "A";
    public static final String Column2MapKey = "B";
```

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

```
}
```

```
    @Passer autre
    public void start(Stage stage) { Scene scene
        = new Scene(new Group()); stage.setTitle("Exemple
        de vue de tableau"); stage.setWidth(300);
        stage.setHeight(500);
```

```
        label final label = new Label("Identifiants étudiants");
        label.setFont(nouvelle police("Arial", 20));
```

```
        TableColumn<Map, String> firstDataColumn = new TableColumn<>("Class A");
        TableColumn<Map, String> secondDataColumn = new TableColumn<>("Classe B");
```

```
        firstDataColumn.setCellValueFactory(new MapValueFactory(Column1MapKey));
        firstDataColumn.setMinWidth(130); secondDataColumn.setCellValueFactory(new
        MapValueFactory(Column2MapKey)); secondDataColumn.setMinWidth(130);
```

```
        TableView table_view = new TableView<>(generateDataInMap());
```

```
        table_view.setEditable(true);
        table_view.getSelectionModel().setCellSelectionEnabled(true);
        table_view.getColumns().addAll(firstDataColumn, secondDataColumn);
```

## Ajout de cartes de données à la table

```

Rappel<TableColumn<Map, String>, TableCell<Map, String>>
cellFactoryForMap = new Callback<TableColumn<Map, String>,
TableCell<Carte, Chaîne>>() {
    @Passer outre
    public TableCell call(TableColumn p) {
        return new TextFieldTableCell(new StringConverter() {
            @Passer outre
            public String toString(Object t) { return
                t.toString();
            }
            @Passer outre
            public Object fromString(String string) { chaîne de retour;
            }
        });
    }
};

firstDataColumn.setCellFactory(cellFactoryForMap);
secondDataColumn.setCellFactory(cellFactoryForMap);

VBox finale vbox = nouvelle VBox();

vbox.setSpacing(5);
vbox.setPadding(nouveaux inserts(10, 0, 0, 10));
vbox.getChildren().addAll(label, table_view);

((Groupe) scene.getRoot()).getChildren().addAll(vbox);

stage.setScene(scene);

spectacle();
}

private ObservableList<Map> generateDataInMap() {
    entier max = 10;
    ObservableList<Carte> allData = FXCollections.observableArrayList(); pour (int je = 1; je <
    max; je++) {
        Map<String, String> dataRow = new HashMap<>();

        valeur de chaîne1 = "A" + i;
        valeur de chaîne2 = "B" + i;

        dataRow.put (Column1MapKey, valeur1);
        dataRow.put (Column2MapKey, valeur2);

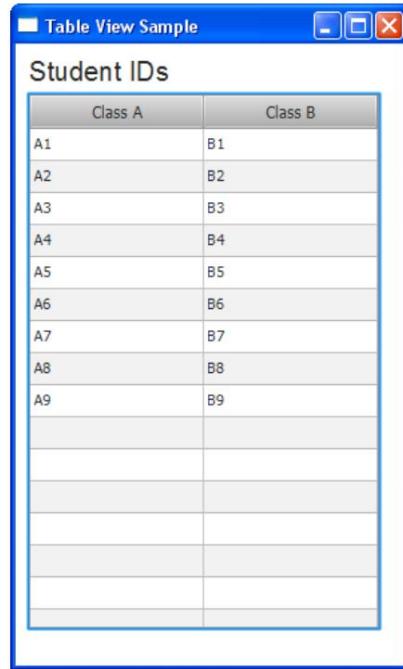
        allData.add (dataRow);
    }
    retourner toutes les données;
}
}

```

La classe MapValueFactory implémente l'interface Callback et est conçue spécifiquement pour être utilisée dans les fabriques de cellules des colonnes de table. Dans l' [exemple 12–12](#), la carte de hachage dataRow présente une seule ligne dans l'objet TableView. La carte a deux clés String : Column1MapKey et Column2MapKey pour mapper les valeurs correspondantes dans les première et deuxième colonnes. La méthode setCellValueFactory appelée pour les colonnes de table les remplit avec des données correspondant à une clé particulière, de sorte que la première colonne contient des valeurs qui correspondent à la clé "A" et la deuxième colonne contient les valeurs qui correspondent à la clé "B".

Lorsque vous compilez et exécutez cette application, elle produit la sortie illustrée à la [Figure 12–9](#).

Figure 12–9 TableView avec données cartographiques



**Documentation relative à l'API**

- ÿ TableView
- ÿ TableColonne
- ÿ TableCell
- ÿ Champ de texte
- ÿ TextFieldTableCell
- ÿ MapValueFactory
- ÿ Bouton

Ajout de cartes de données à la table

---

# 13

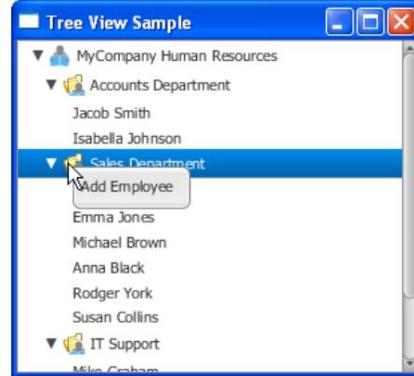
## 13Arborescence

Dans ce chapitre, vous apprendrez à créer des arborescences dans votre application JavaFX, à ajouter des éléments aux arborescences, à traiter des événements et à personnaliser les cellules de l'arborescence en implémentant et en appliquant des fabriques de cellules.

La classe TreeView du package javafx.scene.control fournit une vue des structures hiérarchiques. Dans chaque arbre, l'objet le plus élevé de la hiérarchie est appelé la «racine». La racine contient plusieurs éléments enfants, qui peuvent également avoir des enfants. Un élément sans enfants est appelé "feuille".

[La Figure 13-1 montre une capture d'écran d'une application avec une arborescence.](#)

**Figure 13-1 Exemple d'arborescence**



### Création d'arborescences

Lorsque vous créez une arborescence dans vos applications JavaFX, vous devez généralement instancier la classe TreeView, définir plusieurs objets TreeItem, faire de l'un des éléments de l'arborescence la racine, ajouter la racine à l'arborescence et d'autres éléments de l'arborescence à la racine.

Vous pouvez accompagner chaque élément de l'arborescence d'une icône graphique en utilisant le constructeur correspondant de la classe TreeItem ou en appelant la méthode setGraphic. La taille recommandée pour les icônes est de 16x16, mais en fait, n'importe quel objet Node peut être défini comme icône et il sera entièrement interactif.

[L'exemple 13-1](#) est une implémentation d'une arborescence simple avec la racine et cinq feuilles.

#### Exemple 13-1 Création d'une arborescence

```
importer javafx.application.Application;
importer javafx.scene.Node;
```

```
importer javafx.scene.Scene;
importer javafx.scene.control.TreeItem;
importer javafx.scene.control.TreeView;
importer javafx.scene.image.ImageView;
importer javafx.scene.image.Image;
importer javafx.scene.layout.StackPane;
importer javafx.stage.Stage;

la classe publique TreeViewSample étend l'application {

    rootIcon du nœud final privé = new ImageView(
        nouvelle Image(getClass().getResourceAsStream("folder_16.png"))
    );

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

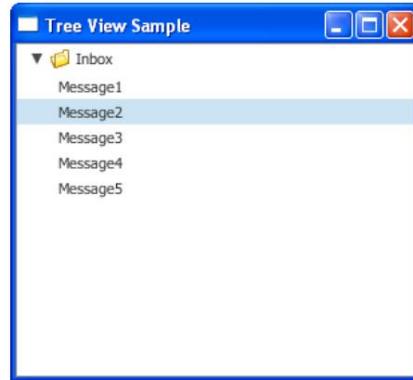
    @Passer autre
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Échantillon d'arborescence");

        TreeItem<String> rootItem = new TreeItem<String> ("Inbox", rootIcon);
        rootItem.setExpanded(true);
        pour (int je = 1; je < 6; je++) {
            TreeItem<String> item = new TreeItem<String> ("Message" + i);
            rootItem.getChildren().add(item);
        }
        TreeView<String> tree = new TreeView<String> (rootItem); StackPane racine = nouveau
        StackPane();
        root.getChildren().add(arbre);
        primaryStage.setScene(nouvelle scène(racine, 300, 250));
        primaryStage.show();
    }
}
```

Tous les éléments de l'arborescence créés dans la boucle for sont ajoutés à l'élément racine en appelant les méthodes getChildren et add. Vous pouvez également utiliser la méthode addAll au lieu de la méthode add pour inclure simultanément tous les éléments d'arborescence précédemment créés.

Vous pouvez spécifier la racine de l'arborescence dans le constructeur de la classe TreeView lorsque vous créez un nouvel objet TreeView, comme illustré dans l' [exemple 13–1](#), ou vous pouvez la définir en appelant la méthode setRoot de la classe TreeView.

La méthode setExpanded appelée sur l'élément racine définit l'apparence initiale de l'élément de l'arborescence. Par défaut, toutes les instances de TreeItem sont réduites et doivent être développées manuellement si nécessaire. Transmettez la valeur true à la méthode setExpanded, de sorte que l'élément de l'arborescence racine semble développé au démarrage de l'application, comme illustré à la [Figure 13–2](#).

**Figure 13–2 Arborescence avec cinq éléments d'arborescence**

L'[exemple 13–1](#) crée une arborescence simple avec les éléments String. Cependant, une arborescence peut contenir des éléments de différents types. Utilisez la notation générique suivante du constructeur TreeItem pour définir des données spécifiques à l'application représentées par un élément d'arbre : TreeItem<T> (valeur T). La valeur T peut spécifier n'importe quel objet, tel que des contrôles d'interface utilisateur ou des composants personnalisés.

Contrairement à la classe TreeView, la classe TreeItem n'étend pas la classe Node.

Par conséquent, vous ne pouvez pas appliquer d'effets visuels ni ajouter de menus aux éléments de l'arborescence.

Utilisez le mécanisme de fabrique de cellules pour surmonter cet obstacle et définissez autant de comportements personnalisés pour les éléments de l'arborescence que votre application l'exige.

## Mise en œuvre des usines cellulaires

Le mécanisme de fabrique de cellules est utilisé pour générer des instances TreeCell pour représenter un seul TreeItem dans le TreeView. L'utilisation de fabriques de cellules est particulièrement utile lorsque votre application fonctionne avec une quantité excessive de données modifiées dynamiquement ou ajoutées à la demande.

Considérez une application qui visualise les données des ressources humaines d'une entreprise donnée et permet aux utilisateurs de modifier les détails des employés et d'ajouter de nouveaux employés.

L'[exemple 13–2](#) crée la classe Employee et organise les employés en groupes selon leurs départements.

### Exemple 13–2 Création d'un modèle de l'arborescence des ressources humaines

```
importer java.util.Arrays;
importer java.util.List;
importer javafx.application.Application;
importer javafx.scene.Node;
importer javafx.scene.Scene;
importer javafx.scene.control.TreeItem;
importer javafx.scene.control.TreeView;
importer javafx.scene.image.Image;
importer javafx.scene.image.ImageView;
importer javafx.scene.paint.Color;
importer javafx.stage.Stage;

importer javafx.beans.property.SimpleStringProperty;
importer javafx.scene.layout.VBox;

la classe publique TreeViewSample étend l'application {
```

```

rootIcon du nœud final privé = new
    ImageView(new Image(getClass().getResourceAsStream("root.png"))); Image finale privée depIcon =
nouvelle Image(getClass().getResourceAsStream("department.png"));

List<Employee> employee = Arrays.<Employee>asList(
    nouvel employé("Ethan Williams", "service des ventes"), nouvel employé("Emma
    Jones", "service des ventes"), nouvel employé("Michael Brown", "service des
    ventes"), nouvel employé("Anna Black", « Service des ventes »), nouvel employé
    (« Rodger York », « Service des ventes »), nouvel employé (« Susan Collins », «
    Service des ventes »), nouvel employé (« Mike Graham », « Support informatique
    »), nouvel employé ("Judy Mayer", "IT Support"), nouvel employé ("Gregory Smith",
    "IT Support"), nouvel employé ("Jacob Smith", "Accounts Department"), nouvel
    employé ("Isabella Johnson", "Accounts Département"));

TreeItem<String> rootNode =
    new TreeItem<String>("MyCompany Human Resources", rootIcon);

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

}

@Passer outre
public void start(Stage stage)
{ rootNode.setExpanded(true); pour (Employé
employé : employés) {
    TreeItem<String> empLeaf = new TreeItem<String>(employee.getName()); booléen trouvé = faux;

    for (TreeItem<String> depNode : rootNode.getChildren()) {
        if (depNode.getValue().contentEquals(employee.getDepartment())){
            depNode.getChildren().add(empLeaf); trouvé = vrai;
            Pause;

        }
    }
    si trouvé) {
        TreeItem<String> depNode = new TreeItem<String>(
            employé.getDepartment(), nouvelle
            ImageView(depIcon)
        );
        rootNode.getChildren().add(depNode);
        depNode.getChildren().add(empLeaf);
    }
}
}

stage.setTitle("Échantillon d'arborescence"); Boîte VBox
= nouvelle VBox(); scène finale scène = nouvelle scène
(boîte, 400, 300); scène.setFill(Color.LIGHTGRAY);

TreeView<String> treeView = new TreeView<String>(rootNode);

box.getChildren().add(treeView);
stage.setScene(scene); spectacle();

}

public static class Employé {

```

```
nom final privé de SimpleStringProperty;
département SimpleStringProperty final privé;

private Employee(String name, String department) {
    this.name = new SimpleStringProperty(nom);
    this.department = new SimpleStringProperty(department);
}

chaîne publique getName() {
    return name.get();
}

public void setName(String fName) {
    nom.set(fName);
}

chaîne publique getDepartment() {
    renvoie département.get();
}

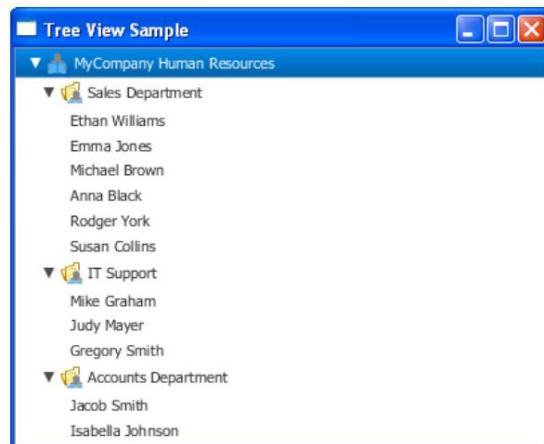
public void setDepartment(String fName) {
    département.set(fName);
}

}
```

Chaque objet Employee de l'exemple 13-2 possède deux propriétés : name et department. Les objets TreeItem correspondant aux employés sont appelés feuilles d'arbre, tandis que les éléments d'arbre correspondant aux départements sont appelés éléments d'arbre avec enfants. Le nom du nouveau service à créer est extrait d'un objet Employee en appelant la méthode getDepartment.

Lorsque vous compilez et exéquez cette application, elle crée la fenêtre illustrée à la Figure 13-3.

**Figure 13–3** Liste des employés dans l'exemple d'application arborescente



Avec l' [exemple 13–2](#), vous pouvez prévisualiser l'arborescence et ses éléments, mais vous ne pouvez pas modifier les éléments existants ni ajouter de nouveaux éléments. L'[exemple 13–3](#) montre une version modifiée de l'application avec la fabrique de cellules implémentée. L'application modifiée vous permet de changer le nom d'un employé.

**Exemple 13-3 Implémentation d'une fabrique de cellules**

```

importer java.util.Arrays; importer
java.util.List; importer
javafx.application.Application; import
javafx.event.EventHandler; importer javafx.scene.Node;
importer javafx.scene.Scene; importer
javafx.scene.control.TextField; importer
javafx.scene.control.TreeCell; importer
javafx.scene.control.TreeItem; importer
javafx.scene.control.TreeView; importer
javafx.scene.image.Image; importer
javafx.scene.image.ImageView; importer
javafx.scene.input.KeyCode; importer
javafx.scene.input.KeyEvent; importer
javafx.scene.paint.Color; importer javafx.stage.Stage;
importer javafx.util.Callback;
importer javafx.beans.property.SimpleStringProperty; importer
javafx.scene.layout.VBox;

la classe publique TreeViewSample étend l'application {

    rootIcon du nœud final privé = new
        ImageView(new Image(getClass().getResourceAsStream("root.png"))); Image finale privée depIcon
    =
        nouvelle Image(getClass().getResourceAsStream("department.png"));
    List<Employee> employee = Arrays.<Employee>asList(
        nouvel employé("Ethan Williams", "Service des ventes"), nouvel
        employé("Emma Jones", "Service des ventes"), nouvel employé("Michael
        Brown", "Service des ventes"), nouvel employé("Anna Black", « Service
        des ventes »), nouvel employé (« Rodger York », « Service des ventes
        »), nouvel employé (« Susan Collins », « Service des ventes »), nouvel
        employé (« Mike Graham », « Support informatique »), nouvel employé
        ("Judy Mayer", "IT Support"), nouvel employé ("Gregory Smith", "IT
        Support"), nouvel employé ("Jacob Smith", "Accounts Department"),
        nouvel employé ("Isabella Johnson", "Accounts Département"));

    TreeItem<String> rootNode =
        new TreeItem<String>("MyCompany Human Resources", rootIcon);

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

    @Passer outre
    public void start(Stage stage)
    { rootNode.setExpanded(true); pour
        (Employé employé : employés) {
            TreeItem<String> empLeaf = new TreeItem<String>(employee.getName()); booléen trouvé = faux;
            for (TreeItem<String> depNode : rootNode.getChildren()) {

                if (depNode.getValue().contentEquals(employee.getDepartment())){
                    depNode.getChildren().add(empLeaf); trouvé =
                    vrai; Pause;

                }
            }
        }
    }
}

```

```

        si trouvé) {
            TreelItem<String> depNode = new TreelItem<String>(
                employé.getDepartment(), nouvelle
                ImageView(depIcon)
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }
    }

    stage.setTitle("Échantillon d'arborescence"); Boîte VBox
    = nouvelle VBox(); scène finale scène = nouvelle scène
    (boîte, 400, 300); scene.setFill(Color.LIGHTGRAY);

    TreeView<String> treeView = new TreeView<String>(rootNode); treeView.setEditable(true);

    treeView.setCellFactory(new Callback<TreeView<String>, TreeCell<String>>(){
        @Passer outre
        public TreeCell<String> call(TreeView<String> p) { return new
            TextFieldTreeCellImpl();
        }
    });

    box.getChildren().add(treeView);
    stage.setScene(scene); spectacle();

}

classe finale privée TextFieldTreeCellImpl étend TreeCell<String> {

    champ de texte privé TextFieldy;

    public TextFieldTreeCellImpl() { }

    @Passer outre
    public void startEdit() { super.startEdit();

        si (textField == null) {
            createTextField();
        }
        setText(null);
        setGraphic(textField);
        textField.selectAll();
    }

    @Passer outre
    public void cancelEdit()
    { super.cancelEdit();
        setText((String) getItem());
        setGraphic(getTreeItem().getGraphic());
    }

    @Passer outre
    public void updateItem (élément de chaîne, booléen vide) { super.updateItem
        (élément, vide);

        si (vide) {

```

```

        setText(null);
        setGraphic(null); }
    autre {
        if (isEditing()) { if
            (textField != null) {
                textField.setText(getString());
            }
            setText(null);
            setGraphic(textField); } autre {
                setText(getString());
                setGraphic(getTreeItem().getGraphic());
            }
        }
    }

privé void createTextField() { textField = new
    TextField(getString()); textField.setOnKeyReleased(new
    EventHandler<KeyEvent>() {

        @Passer autre
        public void handle(KeyEvent t) {
            if (t.getCode() == KeyCode.ENTER) {
                commitEdit(textField.getText());
            } else if (t.getCode() == KeyCode.ESCAPE) { cancelEdit();
                }
            }
        });
    }

chaîne privée getString() {
    retourner getItem() == null ? "" : getItem().toString();
}
}

public static class Employé {

    nom final privé de SimpleStringProperty; département
    SimpleStringProperty final privé;

    private Employee(String name, String department) { this.name = new
        SimpleStringProperty(name); this.department = new
        SimpleStringProperty(department);
    }

    public String getName() { return
        name.get();
    }

    public void setName(String fName) {
        nom.set(fName);
    }

    public String getDepartment() { return
        department.get();
    }

    public void setDepartment(String fName)
        { department.set(fName);
    }
}

```

```

        }
    }
}

```

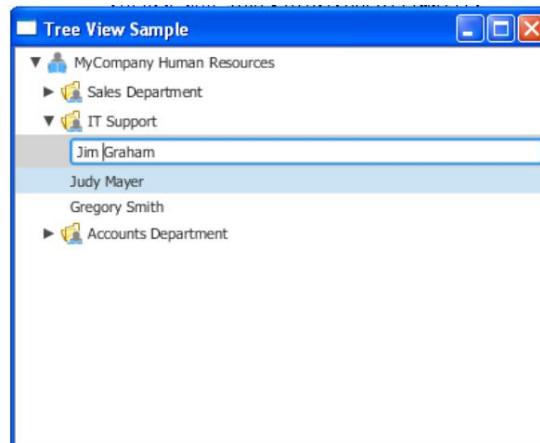
La méthode setCellFactory appelée sur l'objet treeView remplace l'implémentation TreeCell et redéfinit les éléments de l'arborescence comme spécifié dans la classe TextFieldTreeCellImpl.

La classe TextFieldTreeCellImpl crée un objet TextField pour chaque élément de l'arborescence et fournit les méthodes pour traiter les événements d'édition.

Notez que vous devez appeler explicitement la méthode setEditable(true) dans l'arborescence pour permettre la modification de tous ses éléments.

Compilez et exécutez l'application de l' [exemple 13–3](#). Essayez ensuite de cliquer sur les employés dans l'arborescence et de changer leurs noms. La [Figure 13–4](#) capture le moment de la modification d'un élément de l'arborescence dans le service de support informatique.

**Figure 13–4 Modification du nom d'un employé**



### Ajout de nouveaux éléments d'arborescence à la demande

Modifiez l'exemple d'application Tree View afin qu'un représentant des ressources humaines puisse ajouter de nouveaux employés. Utilisez les lignes de code en gras de l' [exemple 13–4](#) comme référence. Ces lignes ajoutent un menu contextuel aux éléments de l'arborescence correspondant aux départements. Lorsque l'élément de menu Ajouter un employé est sélectionné, le nouvel élément d'arborescence est ajouté en tant que feuille au service actuel.

Utilisez la méthode isLeaf pour faire la distinction entre les éléments de l'arborescence des services et les éléments de l'arborescence des employés.

#### Exemple 13–4 Ajout de nouveaux éléments d'arborescence

```

importer java.util.Arrays;
importer java.util.List;
importer javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;

```

Ajout de nouveaux éléments d'arborescence à la demande

```

importer javafx.scene.control.TreeView; importer
javafx.scene.image.Image; importer
javafx.scene.image.ImageView; importer
javafx.scene.input.KeyCode; importer
javafx.scene.input.KeyEvent; importer
javafx.scene.paint.Color; importer javafx.stage.Stage;
importer javafx.util.Callback;
```

```

importer javafx.beans.property.SimpleStringProperty; importer
javafx.scene.control.ContextMenu; importer javafx.scene.control.MenuItem;
importer javafx.scene.layout.VBox;
```

la classe publique TreeViewSample étend l'application {

```

    rootIcon du nœud final privé = new
        ImageView(new Image(getClass().getResourceAsStream("root.png"))); Image finale privée depIcon
    =
        nouvelle Image(getClass().getResourceAsStream("department.png"));
    List<Employee> employee = Arrays.<Employee>asList(
        nouvel employé("Ethan Williams", "service des ventes"), nouvel
        employé("Emma Jones", "service des ventes"), nouvel employé("Michael
        Brown", "service des ventes"), nouvel employé("Anna Black", « Service
        des ventes »), nouvel employé (« Rodger York », « Service des ventes
        »), nouvel employé (« Susan Collins », « Service des ventes »), nouvel employé
        (« Mike Graham », « Support informatique »), nouvel employé
        ("Judy Mayer", "IT Support"), nouvel employé ("Gregory Smith", "IT
        Support"), nouvel employé ("Jacob Smith", "Accounts Department"),
        nouvel employé ("Isabella Johnson", "Accounts Département"));
```

```

TreeItem<String> rootNode =
    new TreeItem<String>("MyCompany Human Resources", rootIcon);

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

}

@Passer autre
public void start(Stage stage)
    { rootNode.setExpanded(true); pour
        (Employé employé : employés) {
            TreeItem<String> empLeaf = new TreeItem<String>(employee.getName()); booléen trouvé = faux;
            for (TreeItem<String> depNode : rootNode.getChildren()) {

                if (depNode.getValue().contentEquals(employee.getDepartment())){
                    depNode.getChildren().add(empLeaf); trouvé =
                    vrai; Pause;

                }
            }
            si trouvé) {
                TreeItem depNode = new TreeItem(employee.getDepartment(), new
                    ImageView(depIcon)
                );
                rootNode.getChildren().add(depNode);
                depNode.getChildren().add(empLeaf);
            }
        }
    }
```

```
stage.setTitle("Échantillon d'arborescence"); Boîte VBox
= nouvelle VBox();
scène finale scène = nouvelle scène (boîte, 400, 300);
scene.setFill(Color.LIGHTGRAY);

TreeView<String> treeView = new TreeView<String>(rootNode); treeView.setEditable(true);

treeView.setCellFactory(new Callback<TreeView<String>, TreeCell<String>(){
    @Passer outre
    public TreeCell<String> call(TreeView<String> p) { return new TextFieldTreeCellImpl(); }

});

box.getChildren().add(treeView);
stage.setScene(scene); spectacle();

}

classe finale privée TextFieldTreeCellImpl étend TreeCell<String> {

    champ de texte privé TextField; menu
    contextuel privé addMenu = new ContextMenu();

    public TextFieldTreeCellImpl() {
        MenuItem addMenuItem = new MenuItem("Ajouter un employé");
        addMenu.getItems().add(addMenuItem); addMenuItem.setOnAction(nouveau
        EventHandler() {
            public void handle(Event t) { Treeltem
                newEmployee = new
                    Treeltem<String>("Nouvel employé");
                    getTreeltem().getChildren().add(newEmployee);
            }
        });
    }

    @Passer outre
    public void startEdit() { super.startEdit();

        si (textField == null) {
            createTextField();
        }
        setText(null);
        setGraphic(textField);
        textField.selectAll();
    }

    @Passer outre
    public void cancelEdit() { super.cancelEdit();

        setText((String) getItem());
        setGraphic(getTreeltem().getGraphic());
    }

    @Passer outre
    public void updateItem (élément de chaîne, booléen vide) { super.updateItem
        (élément, vide);
    }
}
```

```

        si (vide) {
            setText(null);
            setGraphic(null); } autre {

                if (isEditing()) { if (textField !=
                    = null) {
                        textField.setText(getString());
                    }
                    setText(null);
                    setGraphic(textField); } autre {

                        setText(getString());
                        setGraphic(getTreeItem().getGraphic()); si (
                            !getTreeItem().isLeaf()&&getTreeItem().getParent()!= null
                        ){
                            setContextMenu(addMenu);
                        }
                    }
                }
            }

privé void createTextField() { textField = new
    TextField(getString()); textField.setOnKeyReleased(new
    EventHandler<KeyEvent>() {

        @Passer outre
        public void handle(KeyEvent t) {
            if (t.getCode() == KeyCode.ENTER) {
                commitEdit(textField.getText());
            } else if (t.getCode() == KeyCode.ESCAPE) { cancelEdit();

            }
        }
    });
}

}

chaîne privée getString() {
    retourner getItem() == null ? "" : getItem().toString();
}
}

public static class Employé {

    nom final privé de SimpleStringProperty; département
    SimpleStringProperty final privé;

    private Employee(String name, String department) { this.name = new
        SimpleStringProperty(name); this.department = new
        SimpleStringProperty(department);
    }

    public String getName() { return
        name.get();
    }

    public void setName(String fName) {
        nom.set(fName);
    }
}

```

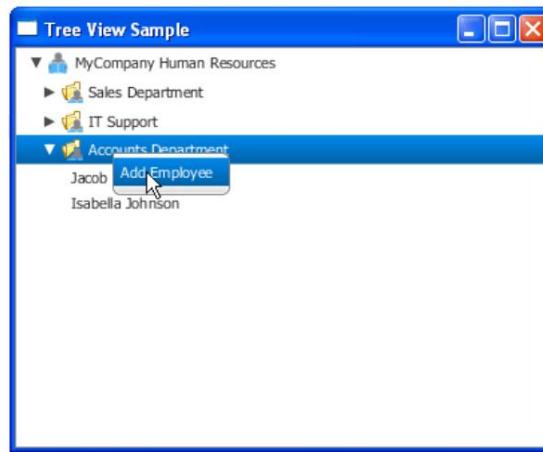
```
        }
    }

    chaîne publique getDepartment() {
        renvoie département.get();
    }

    public void setDepartment(String fName) {
        département.set(fName);
    }
}
```

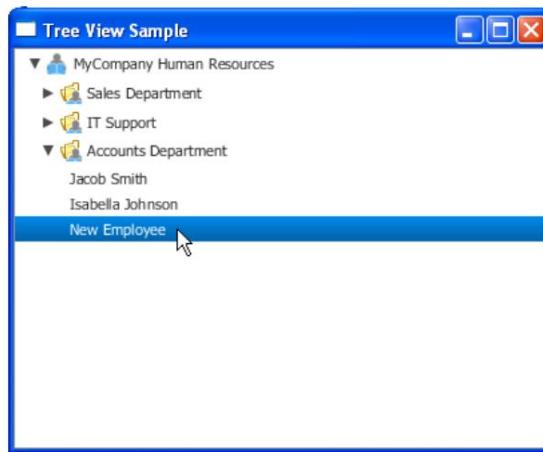
Compiler et exécuter l'application. Sélectionnez ensuite un département dans l'arborescence et faites un clic droit dessus. Le menu contextuel s'affiche, comme illustré à la Figure 13–5.

**Figure 13–5** Menu contextuel pour l'ajout de nouveaux employés



Lorsque vous sélectionnez l'élément de menu Ajouter un employé dans le menu contextuel, le nouvel enregistrement est ajouté au service actuel. La figure 13–6 montre un nouvel élément d'arborescence ajouté au service des comptes.

**Figure 13–6 Employé nouvellement ajouté**



Étant donné que la modification est activée pour les éléments de l'arborescence, vous pouvez remplacer la valeur par défaut "Nouvel employé" par le nom approprié.

**Utilisation des éditeurs de cellules d'arborescence**

À partir de JavaFX SDK 2.2, vous pouvez utiliser les éditeurs de cellules d'arborescence suivants disponibles dans l'API: CheckBoxTreeCell, ChoiceBoxTreeCell, ComboBoxTreeCell, TextFieldTreeCell. Ces classes étendent l'implémentation TreeCell pour restituer un contrôle particulier à l'intérieur de la cellule.

L'[exemple 13–5](#) illustre l'utilisation de la classe CheckBoxTreeCell dans l'interface utilisateur qui construit une structure hiérarchique de cases à cocher.

**Exemple 13–5 Utilisation de la classe CheckBoxTreeCell**

```
importer javafx.application.Application;
importer javafx.scene.Scene;
importer javafx.scene.control.*;
importer javafx.scene.control.cell.CheckBoxTreeCell;
importer javafx.scene.layout.StackPane;
importer javafx.stage.Stage;

la classe publique TreeViewSample étend l'application {

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

    @Passer autre
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Échantillon d'arborescence");

        CheckBoxTreeItem<String> rootItem =
            new CheckBoxTreeItem<String>("Afficher les fichiers sources");
        rootItem.setExpanded(true);

        arbre TreeView final = new TreeView(rootItem); arbre.setEditable(true);

        tree.setCellFactory(CheckBoxTreeCell.<String>forTreeView()); pour (int je = 0; je < 8; je++) {

            final CheckBoxTreeItem<String> checkBoxTreeItem =
                new CheckBoxTreeItem<String>("Sample" + (i+1));
            rootItem.getChildren().add(checkBoxTreeItem);
        }

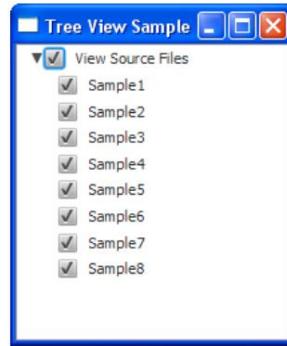
        arbre.setRoot(rootItem);
        arbre.setShowRoot(true);

        StackPane racine = nouveau StackPane();
        root.getChildren().add(arbre);
        primaryStage.setScene(nouvelle scène(racine, 300, 250));
        primaryStage.show();
    }
}
```

L'[exemple 13–5](#) construit l'arborescence en utilisant la classe CheckBoxTreeItem au lieu de TreeItem. La classe CheckBoxTreeItem a été spécialement conçue pour prendre en charge les états sélectionnés, non sélectionnés et indéterminés dans les structures arborescentes. Une instance de CheckBoxTreeItem peut être indépendante ou dépendante. Si une instance de CheckBoxTreeItem est indépendante, toute modification de son état de sélection n'affecte pas ses instances parent et enfant de CheckBoxTreeItem. Par défaut, toutes les instances de CheckBoxTreeItem sont dépendantes.

Compilez et exécutez l'[Exemple 13–5](#), puis sélectionnez l'élément Afficher les fichiers source. Vous devriez voir la sortie illustrée à la [Figure 13–7](#), où tous les éléments enfants sont sélectionnés.

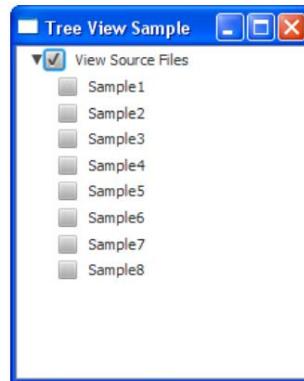
**Figure 13–7 CheckBoxTreeItem dépendant**



Pour rendre une instance de CheckBoxTreeItem indépendante, utilisez la méthode setIndependent :  
rootItem.setIndependent(true);

Lorsque vous exécutez l'application TreeViewSample, son comportement doit changer, comme illustré à la [Figure 13–8](#).

**Figure 13–8 CheckBoxTreeItem indépendant**



#### Documentation relative à l'API

ÿ Arborescence

ÿ ElémentArbre

ÿ TreeCell

ÿ Cellule

ÿ Champ de texte

ÿ CheckBoxTreeCell

ÿ CheckBoxTreeItem



# 14

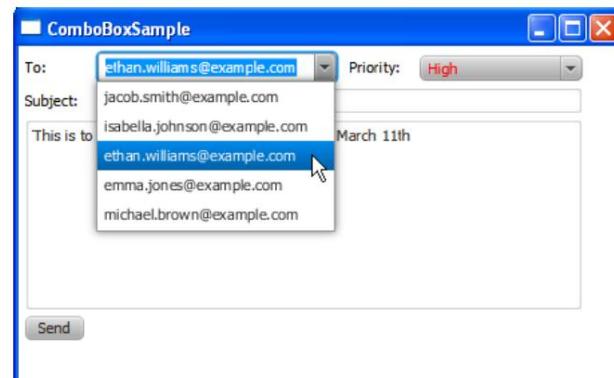
## 14Boîte combinée

Ce chapitre explique comment utiliser les listes déroulantes dans votre application JavaFX. Il traite des zones de liste modifiables et non modifiables, vous apprend à suivre les modifications dans les zones de liste déroulante modifiables et à gérer les événements qui s'y rapportent, et explique comment utiliser des fabriques de cellules pour modifier l'implémentation par défaut d'une zone de liste déroulante.

Une zone de liste déroulante est un élément typique d'une interface utilisateur qui permet aux utilisateurs de choisir l'une des nombreuses options. Une zone de liste déroulante est utile lorsque le nombre d'éléments à afficher dépasse une certaine limite, car elle peut ajouter un défilement à la liste déroulante, contrairement à une zone de choix. Si le nombre d'éléments ne dépasse pas une certaine limite, les développeurs peuvent décider si une zone de liste déroulante ou une zone de choix convient mieux à leurs besoins.

Vous pouvez créer une zone de liste déroulante dans l'application JavaFX en utilisant la classe ComboBox de l'API JavaFX. [La Figure 14–1](#) montre une application avec deux listes déroulantes.

**Figure 14–1 Application avec deux listes déroulantes**



### Création de zones de liste déroulante

Lors de la création d'une zone de liste déroulante, vous devez instancier la classe ComboBox et définir les éléments en tant que liste observable, tout comme les autres contrôles d'interface utilisateur tels que ChoiceBox, ListView et TableView. L'[exemple 14–1](#) définit les éléments dans un constructeur.

#### Exemple 14–1 Cr éation d'une zone de liste déroulante avec une liste observable

```
Options ListeObservable<String> =
    FXCollections.observableArrayList(
        "Option 1",
```

```

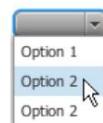
    "Option 2",
    "Option 3"
);
ComboBox final comboBox = new ComboBox(options);

```

Une autre possibilité consiste à créer une zone de liste déroulante en utilisant un constructeur vide et en appelant la méthode `setItems` dessus, comme suit : `comboBox.setItems(options)`;

Lorsque la zone de liste déroulante est ajoutée à la scène de l'application, elle apparaît dans l'interface utilisateur, comme illustré à la [Figure 14–2](#).

**Figure 14–2 Zone combinée avec trois éléments**



A tout moment, vous pouvez compléter la liste des éléments avec de nouvelles valeurs. [Exemple 14–2](#) implémente cette tâche en ajoutant trois éléments supplémentaires au contrôle `comboBox`.

### Exemple 14–2 Ajout d'éléments à une liste déroulante

```

comboBox.getItems().addAll(
    "Option 4",
    "Option 5",
    "Option 6"
);

```

La classe `ComboBox` fournit des propriétés et des méthodes pratiques à utiliser avec les zones de liste déroulante.

Vous pouvez utiliser la méthode `setValue` pour spécifier l'élément sélectionné dans la zone de liste déroulante.

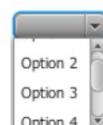
Lorsque vous appelez la méthode `setValue` sur l'objet `ComboBox`, l'élément sélectionné de la propriété `selectionModel` prend cette valeur même si la valeur ne figure pas dans la liste des éléments de la zone de liste déroulante. Si la liste des éléments change ensuite pour inclure cette valeur, l'élément correspondant devient sélectionné.

De même, vous pouvez obtenir la valeur de l'élément sélectionné en appelant le `getValue` méthode. Lorsqu'un utilisateur sélectionne un élément, l'élément sélectionné du modèle de sélection

La propriété et la propriété de valeur de la zone de liste déroulante sont toutes deux mises à jour avec la nouvelle valeur.

Vous pouvez également limiter le nombre de lignes visibles dans la liste déroulante `ComboBox` lorsqu'elle est affichée. La ligne de code suivante permet l'affichage de trois éléments pour le contrôle `comboBox` : `comboBox.setVisibleRowCount(3)`. Suite à l'appel de cette méthode, le nombre de lignes visibles est limité à trois et une barre de défilement apparaît (comme illustré à la [figure 14–3](#)).

**Figure 14–3 Définition du nombre de lignes visibles pour une liste déroulante**



Bien que la classe `ComboBox` ait une notation générique et permette aux utilisateurs de la remplir avec des éléments de différents types, n'utilisez pas `Node` (ou toute sous-classe) comme type. Parce que le

Le concept de graphe scénique implique qu'un seul objet Node peut se trouver à un endroit de la scène de l'application, l'élément sélectionné est supprimé de la liste d'éléments ComboBox. Lorsque la sélection change, l'élément précédemment sélectionné revient dans la liste et la nouvelle sélection est supprimée. Pour éviter cette situation, utilisez le mécanisme de fabrique de cellules et la solution décrite dans la documentation de l'API. Le mécanisme de fabrique de cellules est particulièrement utile lorsque vous devez modifier le comportement initial ou l'apparence de l'objet ComboBox.

L'application ComboBoxSample est conçue pour illustrer l'utilisation des zones de liste déroulante dans une interface de messagerie typique. [L'exemple 14–3](#) crée une telle interface, dans laquelle deux listes déroulantes sont utilisées pour sélectionner le destinataire de l'e-mail et la priorité du message.

#### Exemple 14–3 Création de zones de liste déroulante et ajout de celles-ci à la scène

```
importer javafx.application.Application;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.*;
importer javafx.scene.layout.GridPane;
importer javafx.stage.Stage;

la classe publique ComboBoxSample étend l'application {
    public static void main(String[] args) {
        lancement(arguments);
    }

    Bouton final bouton = nouveau bouton ("Envoyer");
    notification d'étiquette finale = nouvelle étiquette ();
    sujet final TextField = new TextField("");
    texte final de TextArea = new TextArea ("");

    Adresse de chaîne =      " ";

    @Override public void start(Stage stage) {
        stage.setTitle ("ComboBoxSample");
        Scene scene = new Scene(new Group(), 450, 250);

        ComboBox final emailComboBox = new ComboBox();
        emailComboBox.getItems().addAll(
            "jacob.smith@example.com",
            "isabelle.johnson@example.com",
            "ethan.williams@example.com",
            "emma.jones@example.com",
            "michael.brown@example.com"
        );

        ComboBox final priorityComboBox = new ComboBox();
        priorityComboBox.getItems().addAll(
            "Le plus élevé",
            "Haute",
            "Normal",
            "Bas",
            "Le plus bas"
        );

        priorityComboBox.setValue("Normal");

        GridPane grid = new GridPane();
        grille.setVgap(4);
```

```

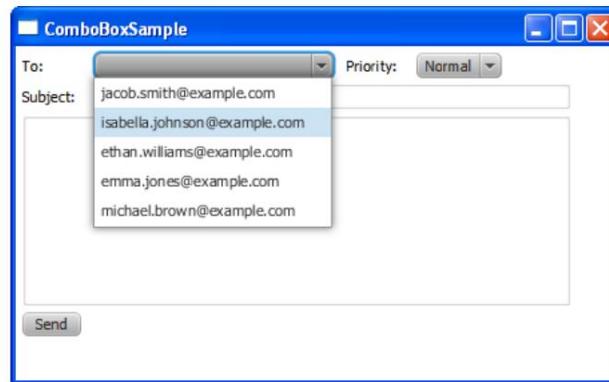
grille.setHgap(10);
grid.setPadding(nouveaux inserts(5, 5, 5, 5));
grid.add(new Label("To: "), 0, 0);
grille.add(emailComboBox, 1, 0);
grid.add(new Label("Priorité : "), 2, 0);
grille.add(priorityComboBox, 3, 0);
grid.add(new Label("Objet : "), 0, 1);
grille.add(sujet, 1, 1, 3, 1); grille.add(texte,
0, 2, 4, 1);
grille.add(bouton, 0, 3);
grille.add(notification, 1, 3, 3, 1);

Racine de groupe = (Groupe)scene.getRoot();
root.getChildren().add(grille);
stage.setScene(scene);
spectacle();
}
}
}

```

Les deux listes déroulantes de l' [exemple 14–3](#) utilisent les méthodes `getItems` et `addAll` pour ajouter des éléments. Lorsque vous compilez et exécutez ce code, il produit la fenêtre d'application illustrée à la [Figure 14–4](#).

**Figure 14–4 Listes déroulantes Destinataire de l'e-mail et Priorité**



## Boîtes combinées modifiables

En règle générale, les applications clientes de messagerie permettent aux utilisateurs de sélectionner des destinataires dans le carnet d'adresses et de saisir une nouvelle adresse. Une zone de liste modifiable s'adapte parfaitement à cette tâche. Utilisez la méthode `setEditable(true)` de la classe `ComboBox` pour rendre une zone de liste déroulante modifiable. Avec la méthode `setPromptText`, vous pouvez spécifier le texte à afficher dans la zone d'édition de la zone de liste déroulante lorsqu'aucune sélection n'est effectuée. Examinez le code modifié de l'application de l' [exemple 14–4](#). Les lignes en gras sont les ajouts faits à l' [exemple 14–3](#).

### Exemple 14–4 Traitement de valeurs nouvellement saisies dans une liste déroulante modifiable

```

importer javafx.application.Application;
importer javafx.beans.value.ChangeListener;
importez javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
importer javafx.geometry.Insets;
importer javafx.scene.Group;

```

```
importer javafx.scene.Scene; importer
javafx.scene.control.*; importer
javafx.scene.layout.GridPane; importer javafx.stage.Stage;

la classe publique ComboBoxSample étend l'application {
    public static void main(String[] args) { launch(args);

    }

    Bouton final bouton = nouveau bouton ("Envoyer");
    notification d'étiquette finale = nouvelle étiquette (); sujet final
    TextField = new TextField(""); texte final de TextArea = new TextArea
    ("");

    Adresse de chaîne =      " "ÿ;

    @Override public void start(Stage stage)
    { stage.setTitle("ComboBoxSample"); Scene scene = new
    Scene(new Group(), 450, 250);

    ComboBox final emailComboBox = new ComboBox();
    emailComboBox.getItems().addAll( "jacob.smith@example.com",
    "isabella.johnson@example.com", "ethan.williams@example.com",
    "emma.jones@example.com", "michael .brown@example.com"

    );
    emailComboBox.setPromptText("Adresse e-mail");
    emailComboBox.setEditable(true);
    emailComboBox.valueProperty().addListener(new ChangeListener<String>() {
        @Passer outre
        public void changé (ObservableValue ov, String t, String t1) {
            adresse = t1ÿ;
        }
    });

    ComboBox final priorityComboBox = new ComboBox();
    priorityComboBox.getItems().addAll( "Le plus haut", "Elevé", "Normal", "Bas",
    "Le plus bas"
    );

    priorityComboBox.setValue("Normal");

    bouton.setOnAction(new EventHandler<ActionEvent>() {
        @Passer outre
        public void handle(ActionEvent e) {
            si (emailComboBox.getValue() != null &&
            !emailComboBox.getValue().toString().isEmpty()){
                notification.setText("Votre message a été envoyé avec succès"
                    + " à "      + adresse)ÿ;
                emailComboBox.setValue(null);
                si (priorityComboBox.getValue() != null && !
                priorityComboBox.getValue().toString().isEmpty()){


```

```

        priorityComboBox.setValue(null);
    }
    sujet.clear();
    text.clear();
}
autre {
    notification.setText("Vous n'avez pas sélectionné de destinataire !");
}
});
GridPane grid = new GridPane();
grille.setVgap(4); grille.setHgap(10);
grid.setPadding(nouveaux inserts(5, 5, 5, 5));
grid.add(new Label("To: "), 0, 0); grille.add(emailComboBox,
1, 0); grid.add(new Label("Priorité : "), 2, 0);
grille.add(priorityComboBox, 3, 0); grid.add(new
Label("Objet : "), 0, 1); grille.add(sujet, 1, 1, 3, 1);
grille.add(texte, 0, 2, 4, 1); grille.add(bouton, 0, 3); grille.add
(notification, 1, 3, 3, 1);

```

```

Racine de groupe = (Groupe)scene.getRoot();
root.getChildren().add(grille); stage.setScene(scene);
spectacle();

```

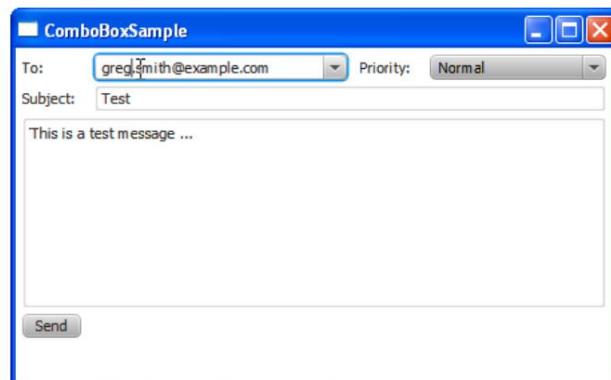
```

}
}
```

Outre la possibilité de modifier emailComboBox, ce fragment de code implémente la gestion des événements pour ce contrôle. La valeur nouvellement saisie ou sélectionnée est stockée dans la variable d'adresse. Lorsque les utilisateurs appuient sur le bouton Envoyer, la notification contenant l'adresse e-mail s'affiche.

La [Figure 14–5](#) capture le moment où un utilisateur modifie l'adresse e-mail de Jacob Smith et la remplace par greg.smith@example.com.

**Figure 14–5 Modification d'une adresse e-mail**



Lorsque le bouton Envoyer est enfoncé, tous les contrôles reviennent à leurs états par défaut. Les méthodes clear sont appelées sur les objets TextField et TextArea, et la valeur null

la valeur est définie pour les éléments sélectionnés de la zone de liste déroulante. La [Figure 14–6](#) montre le moment après avoir appuyé sur le bouton Envoyer.

**Figure 14–6 Interface utilisateur après avoir appuyé sur le bouton Envoyer**



## Application des fabriques de cellules aux zones de liste déroulante

Vous pouvez utiliser le mécanisme de fabrique de cellules pour modifier le comportement ou l'apparence par défaut d'une zone de liste déroulante. [L'exemple 14–5](#) crée une fabrique de cellules et l'applique à la zone de liste déroulante des priorités pour mettre en surbrillance les types de priorité avec des couleurs spéciales.

### Exemple 14–5 Implémentation d'une fabrique de cellules pour la zone de liste déroulante prioritaire

```
importer javafx.application.Application; importer
javafx.beans.value.ChangeListener; importez
javafx.beans.value.ObservableValue; import
javafx.event.ActionEvent; import javafx.event.EventHandler;
importer javafx.geometry.Insets; importer javafx.scene.Group;
importer javafx.scene.Scene; importer javafx.scene.control.*;
importer javafx.scene.layout.GridPane; importer
javafx.scene.paint.Color; importer javafx.stage.Stage; importer
javafx.util.Callback;
```

```
la classe publique ComboBoxSample étend l'application {
    public static void main(String[] args) { launch(args); }
```

```
    Bouton final bouton = nouveau bouton ("Envoyer"); notification
    d'étiquette finale = nouvelle étiquette (); sujet final TextField = new
    TextField(""); texte final de TextArea = new TextArea ("");
```

```
    Adresse de chaîne = " ";
```

```
@Override public void start(Stage stage)
{ stage.setTitle("ComboBoxSample"); Scene scene = new
Scene(new Group(), 450, 250);
```

```
    ComboBox final emailComboBox = new ComboBox();
```

```

emailComboBox.getItems().addAll( "jacob.smith@example.com", "isabella.johnson@example.com", "ethan.williams@example.com", "emma.jones@example.com");
emailComboBox.setPromptText("Adresse e-mail");
emailComboBox.setEditable(true);
emailComboBox.valueProperty().addListener(new ChangeListener<String>() {
    @Override public void changed(ObservableValue ov, String t, String t1)
{
    adresse = t1;
});
PriorityComboBox final priorityComboBox = new ComboBox();
priorityComboBox.getItems().addAll( "Le plus haut", "Elevé", "Normal",
"Bas", "Le plus bas"
);

priorityComboBox.setValue("Normal");
priorityComboBox.setCellFactory(
nouveau Callback<ListView<String>, ListCell<String>>() {
    @Override public ListCell<String> call(ListView<String> param) {
        cellule finale ListCell<String> = new ListCell<String>() {
            {
                super.setPrefWidth(100);
            }
            @Override public void updateItem(String item, boolean empty)
            { super.updateItem(item, empty); si (élément != null) {
                setText(item);
                if (item.contains("High"))
                    { setTextFill(Color.RED);
                }
                sinon si (item.contains("Low")){
                    setTextFill(Color.GREEN);
                }
                autre {
                    setTextFill(Color.BLACK);
                }
            }
            autre {
                setText(null);
            }
        }
        }});
    });
};

bouton.setOnAction(new EventHandler<ActionEvent>() { @Override

```

```
public void handle(ActionEvent e) {
    si (emailComboBox.getValue() != null &&
        !emailComboBox.getValue().toString().isEmpty()){
        notification.setText("Votre message a été envoyé avec succès"
            + " à " + adresse);
        emailComboBox.setValue(null);
        si (priorityComboBox.getValue() != null && !
            priorityComboBox.getValue().toString().isEmpty()){
            priorityComboBox.setValue(null);
        }
        sujet.clear();
        texte.clear();
    }
    autre {
        notification.setText("Vous n'avez pas sélectionné de destinataire !");
    }
}
});;

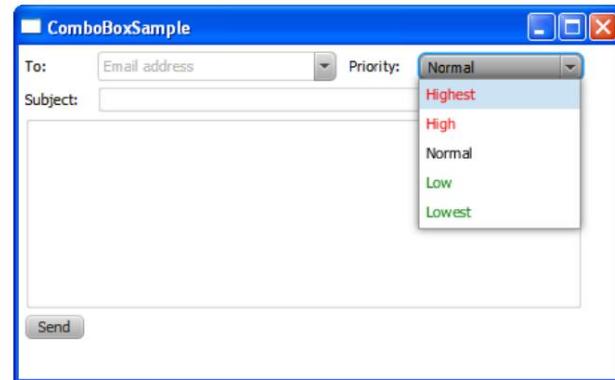
GridPane grid = new GridPane();
grille.setVgap(4);
grille.setHgap(10);
grid.setPadding(nouveaux inserts(5, 5, 5, 5));
grid.add(new Label("To: "), 0, 0);
grille.add(emailComboBox, 1, 0);
grid.add(new Label("Priorité : "), 2, 0);
grille.add(priorityComboBox, 3, 0);
grid.add(new Label("Objet : "), 0, 1);
grille.add(sujet, 1, 1, 3, 1); grille.add(texte,
0, 2, 4, 1);
grille.add(bouton, 0, 3);
grille.add(notification, 1, 3, 3, 1);

Racine de groupe = (Groupe)scene.getRoot();
root.getChildren().add(grille);
stage.setScene(scene);
spectacle();

}
}
```

La fabrique de cellules produit des objets ListCell. Chaque cellule est associée à un seul élément de zone de liste déroulante. La largeur de chaque élément de la zone de liste déroulante est définie via setPrefWidth méthode. La méthode updateItem définit la couleur rouge pour les éléments High et Highest, la couleur verte pour les éléments Low et Lowest, et laisse l'élément Normal noir.

La [Figure 14–7](#) montre les éléments de la liste déroulante des priorités après l'application de la fabrique de cellules de l' [Exemple 14–5](#) .

**Figure 14–7 Modification de la liste déroulante Priorité**

Vous pouvez encore améliorer l'apparence du contrôle ComboBox en appliquant des styles CSS ou des effets visuels.

**Documentation relative à l'API**

- ÿ ComboBox
- ÿ ComboBoxBase
- ÿ ListView
- ÿ CelluleListe
- ÿ Bouton

# 15

## 15 Séparateur

Ce chapitre explique comment utiliser le séparateur pour organiser les composants de l'interface utilisateur de vos applications JavaFX.

La classe Separator disponible dans l'API JavaFX représente une ligne de séparation horizontale ou verticale. Il sert à diviser les éléments de l'interface utilisateur de l'application et ne produit aucune action. Cependant, vous pouvez lui appliquer un style, lui appliquer des effets visuels ou même l'animer. Par défaut, le séparateur est horizontal. Vous pouvez modifier son orientation à l'aide de la méthode setOrientation.

### Création d'un séparateur

Le fragment de code de l' [exemple 15–1](#) crée un séparateur horizontal et un séparateur vertical.

#### Exemple 15–1 Séparateurs verticaux et horizontaux

```
//Séparateur horizontal
Séparateur séparateur1 = nouveau Séparateur();
//Séparateur vertical
Séparateur separator2 = new Separator();
separator2.setOrientation(Orientation.VERTICAL);
```

La classe Separator est une extension de la classe Node. Par conséquent, le séparateur hérite de toutes les variables d'instance de la classe Node.

En règle générale, les séparateurs sont utilisés pour diviser les groupes de contrôles de l'interface utilisateur. Étudiez le fragment de code illustré dans l' [exemple 15–2](#). Il sépare les cases à cocher du mois de printemps des cases à cocher du mois d'été.

#### Exemple 15–2 Utilisation d'un séparateur entre les catégories de cases à cocher

```
final String[] names = new String[]{"Mars", "Avril", "Mai",
        "Juin Juillet aout"};
CheckBox final[] cbs = new CheckBox[noms.longueur];
séparateur final séparateur = nouveau séparateur();
VBox finale vbox = nouvelle VBox();

for (int i = 0; i < noms.longueur; i++) {
    cbs[i] = new CheckBox(names[i]);
}

séparateur.setMaxWidth(40);
separateur.setAlignment(Pos.CENTER_LEFT);
vbox.getChildren().addAll(cbs);
vbox.setSpacing(5);
```

Ajout de séparateurs à l'interface utilisateur de votre application

```
vbox.getChildren().add(3, séparateur);
```

Lorsque ce fragment de code est ajouté à une application, il produit les contrôles illustrés à la [Figure 15–1](#).

**Figure 15–1 Cases à cocher et séparateur**



Un séparateur occupe tout l'espace horizontal ou vertical qui lui est alloué. La méthode `setMaxWidth` est appliquée pour définir une largeur particulière. L'ensemble `VAlignment` spécifie la position verticale du séparateur dans l'espace de mise en page alloué. De même, vous pouvez définir la position horizontale de la ligne de séparation en appliquant la méthode `setHalignment`.

Dans l' [exemple 15–2](#), le séparateur est ajouté à la boîte verticale à l'aide d'une méthode dédiée `add(index, node)`. Vous pouvez utiliser cette approche dans votre application pour inclure des séparateurs après la création de l'interface utilisateur ou lorsque l'interface utilisateur est modifiée de manière dynamique.

## Ajout de séparateurs à l'interface utilisateur de votre application

Comme mentionné précédemment, les séparateurs peuvent être utilisés pour diviser des groupes de contrôles d'interface utilisateur. Vous pouvez également les utiliser pour structurer une interface utilisateur. Considérez la tâche de rendre les données de prévisions météorologiques illustrées à la [Figure 15–2](#).

**Figure 15–2 Structurer les données de prévisions météorologiques avec des séparateurs**



Pour l'application illustrée à la [Figure 15–2](#), des séparateurs sont utilisés pour diviser les objets `Label` et `ImageView`. Étudiez le code source de cette application présentée dans l' [exemple 15–3](#).

### Exemple 15–3 Utilisation de séparateurs dans une application de prévision météo

```
importer javafx.application.Application;
importer javafx.geometry.Insets;
importer javafx.geometry.Orientation;
importer javafx.geometry.VPos;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.*;
importer javafx.scene.image.Image;
importer javafx.scene.image.ImageView;
importer javafx.scene.layout.GridPane;
```

```
importer javafx.scene.text.Font;
importer javafx.stage.Stage;

la classe publique principale étend l'application {

    Légende de l'étiquette = new Label("Prévisions météo");
    Label vendredi = new Label("Vendredi");
    Label samedi = new Label("Samedi");
    Label dimanche = new Label("Dimanche");

    @Passer outre
    public void start(Stage stage) {
        Racine de groupe = nouveau groupe ();
        Scene scene = new Scene(root, 500, 300);
        stage.setScene(scene);
        stage.setTitle("Échantillon séparateur");

        GridPane grille = new GridPane();
        grille.setPadding(nouveaux inserts(10, 10, 10, 10)); grille.setVgap(2);
        grille.setHgap(5);

        scene.setRoot(grille);

        Image cloudImage = new Image(getClass().getResourceAsStream("cloud.jpg"));
        Image sunImage = new Image(getClass().getResourceAsStream("sun.jpg"));

        caption.setFont(Font.font("Verdana", 20));
        GridPane.setConstraints(légende, 0, 0);
        GridPane.setColumnSpan(légende, 8); grille.getChildren().add(légende);

        séparateur final sepHor = nouveau séparateur();
        sepHor.setAlignment(VPos.CENTER);
        GridPane.setConstraints(sepHor, 0, 1);
        GridPane.setColumnSpan(sepHor, 7); grille.getChildren().add(sepHor);

        vendredi.setFont(Font.font("Verdana", 18));
        GridPane.setConstraints(vendredi, 0, 2);
        GridPane.setColumnSpan(vendredi, 2);
        grille.getChildren().add(vendredi);

        séparateur final sepVert1 = nouveau séparateur();
        sepVert1.setOrientation(Orientation.VERTICAL);
        sepVert1.setAlignment(VPos.CENTER); sepVert1.setPrefHeight(80);
        GridPane.setConstraints(sepVert1, 2, 2); GridPane.setRowSpan(sepVert1,
        2); grille.getChildren().add(sepVert1);

        samedi.setFont(Font.font("Verdana", 18));
        GridPane.setConstraints(samedi, 3, 2);
        GridPane.setColumnSpan(samedi, 2);
        grille.getChildren().add(samedi);

        séparateur final sepVert2 = nouveau séparateur();
        sepVert2.setOrientation(Orientation.VERTICAL);
        sepVert2.setAlignment(VPos.CENTRE); sepVert2.setPrefHeight(80);
```

## Séparateurs de style

```

GridPane.setConstraints(sepVert2, 5, 2);
GridPane.setRowSpan(sepVert2, 2);
grille.getChildren().add(sepVert2);

dimanche.setFont(Font.font("Verdane", 18));
GridPane.setConstraints(dimanche, 6, 2);
GridPane.setColumnSpan(dimanche, 2);
grille.getChildren().add(dimanche);

nuage ImageView final = new ImageView(cloudImage);
GridPane.setConstraints(nuage, 0, 3);
grille.getChildren().add(cloud);

Etiquette finale t1 = nouvelle Etiquette("16");
t1.setFont(Font.font("Verdane", 20));
GridPane.setConstraints(t1, 1, 3);
grille.getChildren().add(t1);

final ImageView sun1 = new ImageView(sunImage);
GridPane.setConstraints(sun1, 3, 3);
grille.getChildren().add(sun1);

Etiquette finale t2 = nouvelle Etiquette("18");
t2.setFont(Font.font("Verdane", 20));
GridPane.setConstraints(t2, 4, 3);
grille.getChildren().add(t2);

final ImageView sun2 = new ImageView(sunImage);
GridPane.setConstraints(sun2, 6, 3);
grille.getChildren().add(sun2);

Etiquette finale t3 = nouvelle Etiquette("20");
t3.setFont(Font.font("Verdane", 20));
GridPane.setConstraints(t3, 7, 3);
grille.getChildren().add(t3);

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

}
}

```

Cette application utilise à la fois des séparateurs horizontaux et verticaux et fait en sorte que les séparateurs s'étendent sur les lignes et les colonnes dans le conteneur GridPane. Dans votre application, vous pouvez également définir la longueur préférée d'un séparateur (largeur pour un séparateur horizontal et hauteur pour un séparateur vertical), afin qu'elle puisse changer dynamiquement lorsque l'interface utilisateur se redimensionne. Vous pouvez également modifier l'apparence visuelle d'un séparateur en appliquant les classes CSS disponibles pour les objets Separator.

## Séparateurs de style

Pour appliquer le même style à tous les séparateurs de l' [exemple 15–3](#), vous créez un fichier CSS (par exemple, controlStyle.css) et enregistrez ce fichier dans le même package que la classe principale de votre application. L' [exemple 15–4](#) illustre les classes CSS que vous pouvez ajouter au fichier controlStyle.

**Exemple 15–4 Utilisation des classes CSS pour styliser les séparateurs**

```
/*controlStyle.css */  
  
.separateur{  
    -fx-background-color: #e79423; -rayon  
    d'arrière-plan fxj: 2;  
}
```

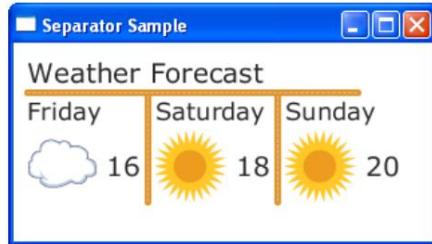
Vous pouvez activer le style de séparateur dans l'application via la méthode `getStylesheets` de la classe `Scene`, comme illustré dans l' [exemple 15–5](#).

**Exemple 15–5 Activation des feuilles de style dans une application JavaFX**

```
scene.getStylesheets().add("separatorsample/controlStyle.css");
```

La [Figure 15–3](#) montre à quoi ressemblent les séparateurs dans les prévisions météorologiques lorsque l'application modifiée est compilée et exécutée.

**Figure 15–3 Séparateurs stylisés**

**Documentation relative à l'API**

ÿ Séparateur

ÿ Spécification CSS JavaFX

Séparateurs de style

---

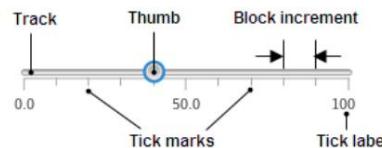
# 16

## 16Curseur

Dans ce chapitre, vous apprendrez à utiliser les curseurs dans vos applications JavaFX pour afficher et interagir avec une plage de valeurs numériques.

Le contrôle Slider se compose d'une piste et d'un pouce déplaçable. Il peut également inclure des graduations et des étiquettes de graduation qui indiquent les valeurs numériques de la plage. [La Figure 16-1](#) montre un curseur typique et indique ses principaux éléments.

**Figure 16-1 Éléments d'un curseur**



### Création d'un curseur

Prenez un moment pour examiner le fragment de code de l' [exemple 16-1](#) qui produit un curseur illustré à la [figure 16-1](#).

#### Exemple 16-1 Création d'un curseur

```
Curseur curseur = nouveau curseur ();
slider.setMin(0);
slider.setMax(100);
slider.setValue (40);
slider.setShowTickLabels(true);
slider.setShowTickMarks(true);
slider.setMajorTickUnit(50);
slider.setMinorTickCount(5);
slider.setBlockIncrement(10);
```

Les méthodes `setMin` et `setMax` définissent respectivement les valeurs numériques minimum et maximum représentées par le curseur. La méthode `setValue` spécifie la valeur actuelle du curseur, qui est toujours inférieure à la valeur maximale et supérieure à la valeur minimale. Utilisez cette méthode pour définir la position du pouce au démarrage de l'application.

Deux méthodes booléennes, `setShowTickMarks` et `setShowTickLabels`, définissent l'apparence visuelle du curseur. Dans l' [exemple 16-1](#), les repères et les étiquettes sont activés.

De plus, l'unité de distance entre les graduations principales est définie sur 50, et le nombre de graduations mineures entre deux graduations majeures est spécifié sur 5. Vous pouvez affecter le

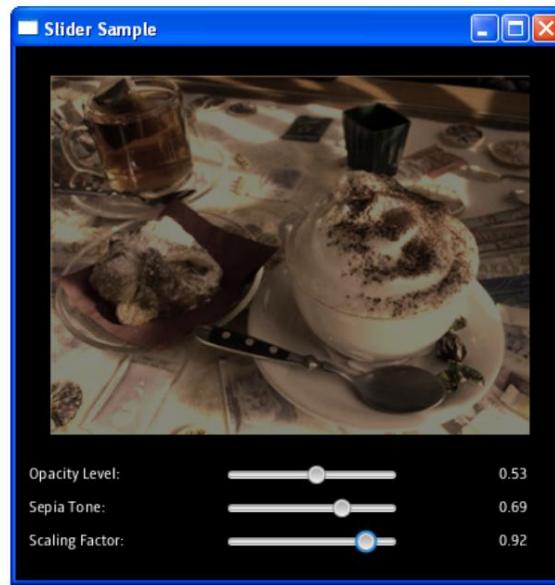
setSnapToTicks sur true pour que la valeur du curseur reste alignée avec les graduations.

La méthode setBlockIncrement définit la distance parcourue par le pouce lorsqu'un utilisateur clique sur la piste. Dans l' [exemple 16–1](#), cette valeur est 10, ce qui signifie que chaque fois qu'un utilisateur clique sur la piste, le pouce se déplace de 10 unités vers l'emplacement du clic.

## Utilisation des curseurs dans les applications graphiques

Examinez maintenant la [Figure 16–2](#). Cette application utilise trois curseurs pour modifier les caractéristiques de rendu d'une image. Chaque curseur ajuste une caractéristique visuelle particulière : niveau d'opacité, valeur de ton sépia ou facteur d'échelle.

**Figure 16–2 Trois curseurs**



L' [exemple 16–2](#) montre le code source de cette application.

### Exemple 16–2 Exemple de curseur

```
importer javafx.application.Application;
importer javafx.beans.value.ChangeListener;
importer javafx.beans.value.ObservableValue;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Label;
importer javafx.scene.control.Slider;
importer javafx.scene.effect.SepiaTone;
importer javafx.scene.image.ImageView;
importer javafx.scene.layout.GridPane;
importer javafx.scene.paint.Color;
importer javafx.stage.Stage;

la classe publique principale étend l'application {

    curseur final opacityLevel = nouveau curseur (0, 1, 1);
```

```
        curseur final sepiaTone = nouveau curseur (0, 1, 1); mise à
        l'échelle finale du curseur = nouveau curseur (0,5, 1, 1)ý;
        Image finale image = nouvelle Image(getClass().getResourceAsStream(
                "cappuccino.jpg")
        );

        final Label opacityCaption = new Label("Opacity Level:"); final Label sepiaCaption =
        new Label("Sepia Toneý:"); final Label scalingCaption = new Label("Facteur
        d'échelle :");

        Étiquette finale opacityValue = nouvelle étiquette(
                Double.toString(opacityLevel.getValue())); Étiquette finale sepiaValue
        = nouvelle étiquette(
                Double.toString(sepiaTone.getValue()));
        étiquette finale scalingValue = nouvelle étiquette (
                Double.toString(scaling.getValue()));

        couleur statique finale textColor = Color.WHITEy; final static
        SepiaTone sepiaEffect = new SepiaTone();

        @Passer outre
        public void start(Stage stage) { Group root =
                new Group(); Scene scene = new
                Scene(root, 600, 400); stage.setScene(scene);
                stage.setTitle("Slider Sample"); scene.setFill(Color.BLACK);

                GridPane grid = new GridPane();
                grid.setPadding(nouveaux inserts(10, 10, 10, 10));
                grille.setVgap(10); grille.setHgap(70);

                final ImageView cappuccino = nouvelle ImageView (image);
                cappuccino.setEffect(sepiaEffect); GridPane.setConstraints(cappuccino,
                0, 0); GridPane.setColumnSpan(cappuccino, 3);
                grille.getChildren().add(cappuccino); scene.setRoot(grille);

                opacityCaption.setTextFill(textColor);
                GridPane.setConstraints(opacityCaption, 0, 1);
                grille.getChildren().add(opacityCaption);

                opacityLevel.valueProperty ().

                cappuccino.setOpacity(new_val.doubleValue());
                opacityValue.setText(String.format("%.2f", new_val));
            }
        });

        GridPane.setConstraints(opacityLevel, 1, 1);
        grille.getChildren().add(opacityLevel);

        opacityValue.setTextFill(textColor);
        GridPane.setConstraints(opacityValue, 2, 1);
        grille.getChildren().add(opacityValue);
```

```

        sepiaCaption.setTextFill(textColor);
        GridPane.setConstraints(sepiaCaption, 0, 2);
        grid.getChildren().add(sepiaCaption);

        sepiaTone.valueProperty().addListener(new ChangeListener<Number>() {
            public void changed(ObservableValue<? extends Number> ov, Number old_val,
                Number new_val) {
                    sepiaEffect.setLevel(new_val.doubleValue());
                    sepiaValue.setText(String.format("%.2f", new_val));
                }
        });
        GridPane.setConstraints(sepiaTone, 1, 2);
        grille.getChildren().add(sepiaTone);

        sepiaValue.setTextFill(textColor);
        GridPane.setConstraints(sepiaValue, 2, 2);
        grid.getChildren().add(sepiaValue);

        scalingCaption.setTextFill(textColor);
        GridPane.setConstraints(scalingCaption, 0, 3);
        grille.getChildren().add(scalingCaption);

        scaling.valueProperty().addListener(new ChangeListener<Number>() { public void
            changed(ObservableValue<? extends Number> ov, Number old_val, Number new_val) {

                cappuccino.setScaleX(new_val.doubleValue());
                cappuccino.setScaleY(new_val.doubleValue());
                scalingValue.setText(String.format("%.2f", new_val));
            }
        });
        GridPane.setConstraints(mise à l'échelle, 1, 3);
        grille.getChildren().add(mise à l'échelle);

        scalingValue.setTextFill(textColor);
        GridPane.setConstraints(scalingValue, 2, 3);
        grille.getChildren().add(scalingValue);

        spectacle();
    }

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

```

La propriété opacity de l'objet ImageView change en fonction de la valeur du premier curseur, nommé opacityLevel. Le niveau de l'effet SepiaTone change lorsque la valeur du curseur sepiaTone change. Le troisième curseur définit le facteur d'échelle de l'image en transmettant aux méthodes setScaleX et setScaleY la valeur actuelle du curseur.

Le fragment de code de l' [exemple 16–3](#) illustre les méthodes qui convertissent la valeur double renvoyée par la méthode getValue de la classe Slider en String. Il applique également le formatage pour rendre la valeur du curseur sous la forme d'un nombre flottant avec deux chiffres après le point.

### Exemple 16–3 Formatage de la valeur du curseur rendu

```
scalingValue.setText((Double.toString(valeur)).format("%.2f", valeur));
```

La prochaine étape pour améliorer l'apparence d'un curseur consiste à lui appliquer des effets visuels ou des styles CSS.

**Documentation relative à l'API**

ÿ Curseur

ÿ Sépia



# 17

## 17Barre de progression et indicateur de progression

Dans ce chapitre, vous découvrirez l'indicateur de progression et la barre de progression, les contrôles de l'interface utilisateur qui visualisent la progression de toutes les opérations dans vos applications JavaFX.

La classe ProgressIndicator et sa sous-classe directe ProgressBar fournissent les capacités pour indiquer qu'une tâche particulière est en cours de traitement et pour détecter la quantité de ce travail qui a déjà été effectuée. Alors que la classe ProgressBar visualise la progression sous la forme d'une barre d'achèvement, la classe ProgressIndicator visualise la progression sous la forme d'un graphique à secteurs changeant dynamiquement, comme illustré à la [Figure 17-1](#).

**Figure 17-1 Barre de progression et indicateur de progression**



### Création de contrôles de progression

Utilisez le fragment de code de l' [exemple 17-1](#) pour insérer les contrôles de progression dans votre application JavaFX.

#### Exemple 17-1 Implémentation de la barre de progression et de l'indicateur de progression

```
ProgressBar pb = nouvelle ProgressBar(0.6);
ProgressIndicator pi = nouveau ProgressIndicator(0.6);
```

Vous pouvez également créer les contrôles de progression sans paramètres en utilisant un constructeur vide. Dans ce cas, vous pouvez affecter la valeur à l'aide de la méthode setProgress.

Parfois, une application ne peut pas déterminer le temps d'exécution complet d'une tâche. Dans ce cas, les contrôles de progression restent en mode indéterminé jusqu'à ce que la durée de la tâche soit déterminée. La [Figure 17-2](#) montre différents états des contrôles de progression en fonction de leur valeur de variable de progression.

**Figure 17–2 Différents états des contrôles de progression**

L'[exemple 17–2](#) montre le code source de l'application illustrée à la [figure 17–2](#).

### Exemple 17–2 Activation de différents états de contrôle de progression

```
importer javafx.application.Application;
importer javafx.geometry.Pos;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Label;
importer javafx.scene.control.ProgressBar;
importer javafx.scene.control.ProgressIndicator;
importer javafx.scene.layout.HBox;
importer javafx.scene.layout.VBox;
importer javafx.stage.Stage;
```

```
la classe publique principale étend l'application {  
  
valeurs finales de Float[] = new Float[] {-1.0f, 0f, 0.6f, 1.0f}; Étiquette finale [] étiquettes =  
nouvelle étiquette [valeurs.longueur]; final ProgressBar[] pbs = new ProgressBar [values.length];  
broches finales ProgressIndicator[] = new ProgressIndicator [values.length]; HBox finale hbs []  
= nouvelle HBox [values.length];
```

```
@Passer autre  
public void start(Stage stage) { Group root = new  
Group(); Scene scene = new Scene(root,  
300, 150); scene.getStylesheets().add("progresssample/  
Style.css"); stage.setScene(scene); stage.setTitle("Contrôles de progression");
```

```
for (int i = 0; i < values.length; i++) {  
    Étiquette finale étiquette = étiquettes[i] = nouvelle étiquette();  
    label.setText("progrès:" + valeurs[i]);  
  
    final ProgressBar pb = pbs[i] = new ProgressBar(); pb.setProgress(values[i]);  
  
    pin finale ProgressIndicator = pins[i] = new ProgressIndicator(); pin.setProgress(values[i]); HBox final hb  
    = hbs[i] = new HBox(); hb.setSpacing(5); hb.setAlignment(Pos.CENTRE); hb.getChildren().addAll(label,  
    pb, pin);  
  
}  
  
VBox finale vb = nouvelle VBox();
```

```

vb.setSpacing(5);
vb.getChildren().addAll(hbs);
scène.setRoot(vb);
spectacle();
}
public static void main(String[] args) { launch(args);

}
}

```

Une valeur positive de la variable de progression entre 0 et 1 indique le pourcentage de progression. Par exemple, 0,4 correspond à 40 %. Une valeur négative pour cette variable indique que la progression est en mode indéterminé. Utilisez la méthode isIndeterminate pour vérifier si le contrôle de progression est en mode indéterminé.

### Indication de la progression dans votre interface utilisateur L'

[exemple 17–2](#) a été initialement simplifié pour rendre tous les états possibles des contrôles de progression. Dans les applications du monde réel, la valeur de progression peut être obtenue via la valeur d'autres éléments de l'interface utilisateur.

Étudiez le code de l' [exemple 17–3](#) pour savoir comment définir les valeurs de la barre de progression et de l'indicateur de progression en fonction de la position du curseur.

#### Exemple 17–3 Réception de la valeur de progression d'un curseur

```

importer javafx.application.Application; importer
javafx.beans.value.ChangeListener<?>; importez
javafx.beans.value.ObservableValue<?>; importer
javafx.geometry.Pos<?>; importer javafx.scene.Group<?>; importer
javafx.scene.Scene<?>; importer javafx.scene.control.ProgressBar<?>;
importer javafx.scene.control.ProgressIndicator<?>; importer
javafx.scene.control.Slider<?>; importer javafx.scene.layout.HBox<?>;
importer javafx.stage.Stage<?>;

```

la classe publique principale étend l'application {

```

@Passer outre
public void start(Stage stage) {
    Racine de groupe = nouveau groupe ();
    Scene scene = new Scene(root);
    stage.setScene(scene);
    stage.setTitle("Contrôles de progression");

    Curseur final slider = new Slider(); slider.setMin(0);
    slider.setMax(50);

```

barre de progression finale pb = nouvelle barre de

progression (0); ProgressIndicator final pi = nouveau ProgressIndicator(0);

```

slider.valueProperty (). setProgress(new_val.doubleValue()/50);

```

```
        }

    });

HBox final hb = new HBox();
hb.setSpacing(5);
hb.setAlignment(Pos.CENTRE);
hb.getChildren().addAll(curseur, pb, pi); scène.setRoot(hb);

spectacle();
}

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

}

}
```

Lorsque vous compilez et exécutez cette application, elle produit la fenêtre illustrée à la [Figure 17–3](#).

Figure 17–3 Indication de la progression définie par un curseur



Un objet ChangeListener détermine si la valeur du curseur est modifiée et calcule la progression de la barre de progression et de l'indicateur de progression afin que les valeurs des contrôles de progression soient comprises entre 0,0 et 1,0.

**Documentation relative à l'API.**

- ÿ Barre de progression
- ÿ Indicateur de progression

# 18

## 18Hyperlien

Ce chapitre décrit le contrôle Hyperlink utilisé pour formater le texte en lien hypertexte.

La classe Hyperlink représente un autre type de contrôle Labeled. [Image 18–1](#) illustre trois états de l'implémentation de lien hypertexte par défaut.

**Figure 18–1 Trois états d'un contrôle de lien hypertexte**

http://example.com	—	unvisited link
	—	link is clicked
http://example.com	—	visited link

### Création d'un lien hypertexte

Le fragment de code qui produit un lien hypertexte est présenté dans l' [Exemple 18–1](#).

#### Exemple 18–1 Lien hypertexte type

```
Lien hypertexte = nouveau lien hypertexte();
lien.setText("http://exemple.com");
lien.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) {
        System.out.println("Ce lien est cliqué");
    }
});
```

La méthode d'occurrence setText définit la légende de texte pour le lien hypertexte. Étant donné que la classe Hyperlink est une extension de la classe Labeled, vous pouvez définir une police et un remplissage de texte spécifiques pour la légende du lien hypertexte. La méthode setOnAction définit l'action spécifique, qui est appelée chaque fois que l'on clique sur le lien hypertexte, de la même manière que cette méthode fonctionne pour le contrôle Button. Dans l' [Exemple 18–1](#), cette action se limite à imprimer la chaîne. Cependant, dans votre application, vous souhaiterez peut-être implémenter des tâches plus courantes.

### Lier le contenu local

L'application de la [Figure 18–2](#) rend les images à partir du répertoire local.

## Lier le contenu local

---

**Figure 18–2 Affichage des images**



Passez en revue le code source de cette application illustrée dans l' [exemple 18–2](#).

### Exemple 18–2 Utilisation d'hyperliens pour afficher des images

```

importer javafx.application.Application; import
javafx.event.ActionEvent; import javafx.event.EventHandler;
import javafx.scene.*; importer javafx.scene.control.*;
importer javafx.scene.image.ImageView; importer
javafx.scene.image.ImageView; importer
javafx.scene.layout.VBox; importer javafx.stage.Stage;

la classe publique principale étend l'application {

    chaîne statique finale [] imageFiles = nouvelle chaîne [] {
        "produit.png",
        "éducation.png",
        "partenaires.png",
        "support.png"
    }

    }jy; légendes finales de chaîne statique [] = nouvelle chaîne [] {
        "Des produits",
        "Éducation",
        "Les partenaires",
        "Soutien"

    }jy; ImageView final sélectionnéImage = new ImageView(); liste finale de
    ScrollPane = new ScrollPane(); lien hypertexte final [] hpls = nouveau lien
    hypertexte [légendes.longueur]jy; image finale [] images = nouvelle image [imageFiles.length]jy;

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

    }

    @Passer outre
    public void start(Stage stage) { Scene scene = new
        Scene(new Group()); stage.setTitle("Exemple de lien
        hypertexte"); stage.setWidth(300); stage.setHeight(200);

        selectedImage.setLayoutX(100);
        selectedImage.setLayoutY(10);

        for (int je = 0; je < captions.length; i++) {
            lien hypertexte final hpl = hpls[i] = nouveau lien hypertexte (légendes[i]);
        }
    }
}

```

```

        Image finale image = images[i] = nouvelle Image(
            getClass().getResourceAsStream(imageFiles[i])
        );
        hpl.setOnAction(new EventHandler<ActionEvent>() {
            @Passer outre
            public void handle(ActionEvent e)
                { selectedImage.setImage(image);
            }
        });
    }

bouton final button = new Button("Actualiser les liens");
bouton.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) { for (int i = 0; i <
        captions.length; i++) { hpis[i].setVisited(false);
        ImageSélectionnée.setImage(null);

    }
}
));
}

VBox vbox = nouvelle VBox();
vbox.getChildren().addAll(hpls);
vbox.getChildren().add(bouton);
vbox.setSpacing(5);

((Groupe) scene.getRoot()).getChildren().addAll(vbox, selectedImage); stage.setScene(scene);
spectacle();
}

}
}

```

Cette application crée quatre objets Hyperlink dans une boucle for. La méthode setOnAction appelée sur chaque lien hypertexte définit le comportement lorsqu'un utilisateur clique sur un lien hypertexte particulier. Dans ce cas, l'image correspondante du tableau images est définie pour la variable selectedImage.

Lorsqu'un utilisateur clique sur un lien hypertexte, il devient visité. Vous pouvez utiliser la méthode setVisited de la classe Hyperlink pour actualiser les liens. Le fragment de code de l' [exemple 18–3](#) accomplit cette tâche.

#### **Exemple 18–3 Actualisation des hyperliens**

```

bouton final button = new Button("Actualiser les liens");
bouton.setOnAction(new EventHandler<ActionEvent>() {
    @Passer outre
    public void handle(ActionEvent e) { for (int i = 0; i <
        captions.length; i++) { hpis[i].setVisited(false);
        ImageSélectionnée.setImage(null);

    }
}
));

```

Lorsque vous cliquez dessus, le bouton Actualiser les liens fait passer tous les hyperliens à l'état non visité, comme illustré à la [Figure 18–3](#).

## Liaison du contenu distant

**Figure 18–3 Hyperliens non visités**

Étant donné que la classe `Hyperlink` est une extension de la classe `Labeled`, vous pouvez spécifier non seulement une légende de texte, mais également une image. L'application fournie dans la section suivante utilise à la fois une légende de texte et une image pour créer des hyperliens et pour charger des pages HTML distantes.

## Liaison du contenu distant

Vous pouvez rendre le contenu HTML dans votre application JavaFX en incorporant le `WebView` navigateur dans la scène d'application. Le composant `WebView` fournit une fonctionnalité de navigation de base sur les pages Web. Il rend les pages Web et prend en charge l'interaction de l'utilisateur, comme la navigation dans les liens et l'exécution de commandes JavaScript.

Étudiez le code source de l'application dans l'[exemple 18–4](#). Il crée quatre hyperliens avec des légendes de texte et des images. Lorsqu'un lien hypertexte est cliqué, la valeur correspondante est transmise en tant qu'URL au navigateur intégré.

### Exemple 18–4 Chargement de pages Web distantes

```
importer javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

la classe publique principale étend l'application {

    chaîne statique finale [] imageFiles = nouvelle chaîne [] {
        "produit.png",
        "éducation.png",
        "partenaires.png",
        "support.png"
    };
    légendes finales de chaîne statique [] = nouvelle chaîne [] {
        "Des produits",
        "Éducation",
        "Les partenaires",
        "Soutien"
    };
}
```

```

URL statiques finales de chaîne[] = nouvelle chaîne[]
    "http://www.oracle.com/us/products/index.html", "http://
    education.oracle.com/", "http://www.oracle.com/partners/index.html" ,
    "http://www.oracle.com/us/support/index.html"

};

Imageview final sélectionnéImage = new Imageview(); lien hypertexte
final [] hpls = nouveau lien hypertexte [légendes.longueur]; image finale [] images
= nouvelle image [imageFiles.length];

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

@Passer outre
public void start(Stage stage) { VBox vbox =
    new VBox(); Scene scene = new
    Scene(vbox); stage.setTitle("Exemple de
    lien hypertexte"); stage.setWidth(570);
    stage.setHeight(550);

    selectedImage.setLayoutX(100);
    selectedImage.setLayoutY(10);

    navigateur WebView final = new WebView(); WebEngine
    final webEngine = navigateur.getEngine();

    for (int je = 0; je < captions.length; i++) {
        lien hypertexte final hpl = hpls[i] = nouveau lien hypertexte (légendes[i]);

        Image finale image = images[i] = new
            Image(getClass().getResourceAsStream(imageFiles[i]));
        hpl.setGraphic(nouvelle Imageview (image));
        hpl.setFont(Font.font("Arial", 14)); chaîne finale url =
        urls[i];
    }

    hpl.setOnAction(new EventHandler<ActionEvent>() {
        @Passer outre
        public void handle(ActionEvent e) { webEngine.load(url);

        }
    });
}

HBox hbox = new HBox();
hbox.getChildren().addAll(hpls);

vbox.getChildren().addAll(hbox, navigateur);
VBox.setVgrow(navigateur, Priority.ALWAYS);

stage.setScene(scene);
spectacle();
}
}

```

Les hyperliens sont créés dans une boucle for similaire à celle de l' [exemple 18-2](#). L'action définie pour un lien hypertexte transmet l'URL correspondante du tableau urls à l'objet WebEngine du navigateur intégré.

## Liaison du contenu distant

Lorsque vous compilez et exécutez cette application, elle produit la fenêtre illustrée à la Figure 18–4.

**Figure 18–4 Chargement de pages depuis le site d'entreprise d'Oracle**



**Documentation relative à l'API**

- ÿ Lien hypertexte
- ÿ Étiqueté
- ÿ Affichage Web
- ÿ Moteur Web

# 19

## 19 Éditeur HTML

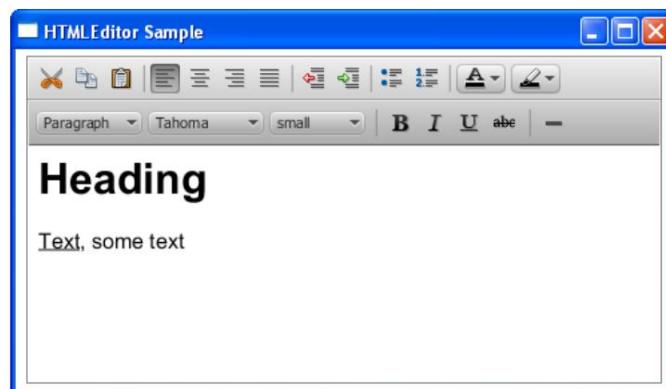
Dans ce chapitre, vous apprendrez à modifier du texte dans vos applications JavaFX à l'aide de l'éditeur HTML intégré.

Le contrôle HTMLEditor est un éditeur de texte enrichi entièrement fonctionnel. Son implémentation est basée sur la fonctionnalité d'édition de documents de HTML5 et inclut les fonctions d'édition suivantes:

- ÿ Mise en forme du texte, y compris les styles gras, italique, souligné et barré
- ÿ Paramètres de paragraphe tels que le format, la famille de polices et la taille de police
- ÿ Couleurs de premier plan et d'arrière-plan
- ÿ Retrait de texte
- ÿ Listes à puces et numérotées
- ÿ Alignement du texte
- ÿ Ajouter une règle horizontale
- ÿ Copier et coller des fragments de texte

[La Figure 19–1](#) montre un éditeur de texte enrichi ajouté à une application JavaFX.

Figure 19–1 Éditeur HTML



La classe HTMLEditor présente le contenu d'édition sous la forme d'une chaîne HTML. Par exemple, le contenu saisi dans l'éditeur de la [Figure 19–1](#) est présenté par la chaîne suivante: "<html><head></head><body contenteditable='true'><h1>Heading</h1><div><u>Texte</u>, du texte</div></body></html>."

## Ajout d'un éditeur HTML

Étant donné que la classe HTMLEditor est une extension de la classe Node, vous pouvez appliquer des effets visuels ou des transformations à ses instances.

## Ajout d'un éditeur HTML

Comme tout autre contrôle d'interface utilisateur, le composant HTMLEditor doit être ajouté à la scène pour qu'il puisse apparaître dans votre application. Vous pouvez l'ajouter directement à la scène comme illustré dans l'[exemple 19–1](#) ou via un conteneur de mise en page comme dans d'autres exemples.

### Exemple 19–1 Ajout d'un éditeur HTML à une application JavaFX

```
importer javafx.application.Application;
importer javafx.scene.Scene;
importer javafx.scene.web.HTMLEditor;
importer javafx.stage.Stage;

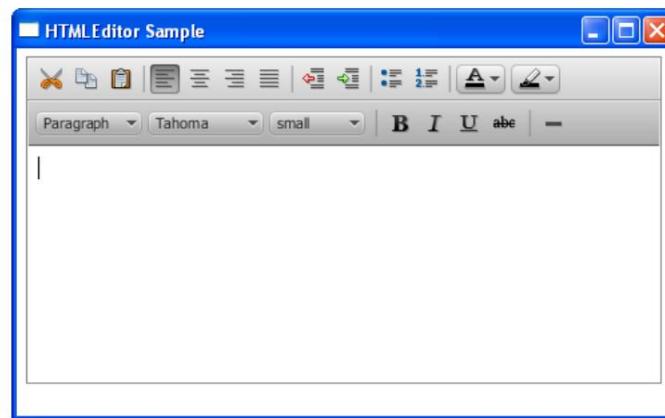
la classe publique HTMLEditorSample étend l'application {

    @Passer autre
    public void start(Stage stage) {
        stage.setTitle("HTMLEditor Sample");
        stage.setWidth(400);
        stage.setHeight(300); ÉditeurHTML
        final ÉditeurHTML = new ÉditeurHTML();
        htmlEditor.setPrefHeight(245);
        Scene scene = new Scene(htmlEditor);
        stage.setScene(scene);
        spectacle();
    }

    public static void main(String[] args) {
        lancement(arguments);
    }
}
```

La compilation et l'exécution de ce fragment de code produisent la fenêtre illustrée à la [Figure 19–2](#).

**Figure 19–2 Vue initiale du composant HTMLEditor**



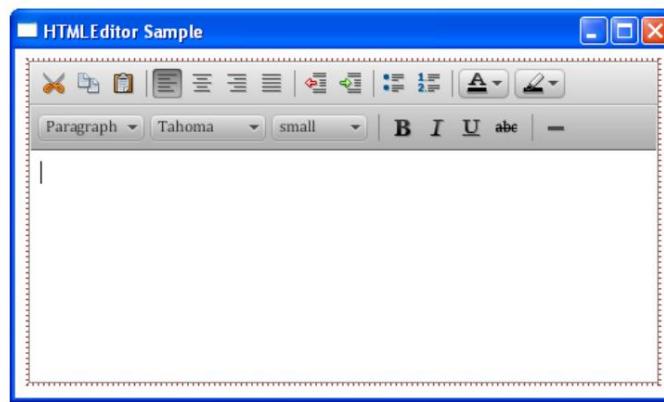
Les barres d'outils de mise en forme sont fournies dans l'implémentation du composant. Vous ne pouvez pas basculer leur visibilité. Cependant, vous pouvez toujours personnaliser l'apparence de l'éditeur en appliquant un style CSS, comme illustré dans l'[exemple 19–2](#).

**Exemple 19–2 Stylisation de l'éditeur HTML**

```
htmlEditor.setStyle( "fx-
    font: 12 cambria;" +
    "-fx-border-color: marron;" +
    "-fx-border-style: pointillé;" + "-fx-
    border-width: 2px;" );
};
```

Lorsque cette ligne de code est ajoutée à l' [Exemple 19–1](#), l'éditeur change, comme illustré à la [Figure 19–3](#).

**Figure 19–3 Affichage alternatif du composant HTMLEditor**



Le style appliqué modifie la bordure du composant et la police des barres d'outils de mise en forme.

La classe HTMLEditor fournit une méthode pour définir le contenu qui apparaît dans la zone d'édition lorsque l'application démarre. Utilisez la méthode `setHtmlText`, comme illustré dans l' [exemple 19–3](#) pour définir le texte initial de l'éditeur.

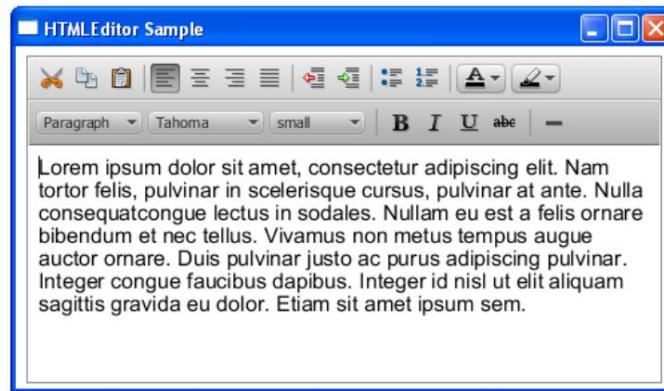
**Exemple 19–3 Définition du contenu du texte**

```
private final String INITIAL_TEXT = "<html><body>Tard la douleur elle-même Aucun football n'est un
football à décorer On n'a pas le temps de vivre la peur de l'auteur +"ornare.
```

+ "Effectué pour le goût du repas.

```
htmlEditor.setHtmlText(INITIAL_TEXT);
```

La figure 19–4 illustre l'éditeur de texte avec le texte défini à l'aide de la méthode `setHTMLText`.

**Figure 19–4 Éditeur HTML avec le contenu textuel prédéfini**

Vous pouvez utiliser les balises HTML de cette chaîne pour appliquer une mise en forme spécifique au contenu initialement rendu.

#### **Utilisation d'un éditeur HTML pour créer l'interface utilisateur**

Vous pouvez utiliser le contrôle HTMLEditor pour implémenter des interfaces utilisateur (UI) typiques dans vos applications JavaFX. Par exemple, vous pouvez implémenter des services de messagerie instantanée, des clients de messagerie ou même des systèmes de gestion de contenu.

L'[exemple 19–4](#) présente l'interface utilisateur d'une fenêtre de composition de message que vous pouvez trouver dans de nombreuses applications client de messagerie.

##### **Exemple 19–4 Éditeur HTML ajouté à l'interface utilisateur du client de messagerie**

```
importer javafx.application.Application; importer
javafx.collections.FXCollections; importer
javafx.geometry.Insets; importer javafx.geometry.Pos;
importer javafx.scene.Group; importer
javafx.scene.Scene; importer javafx.scene.control.*;
importer javafx.scene.layout.GridPane; importer
javafx.scene.layout.VBox; importer
javafx.scene.web.HTMLElement; importer
javafx.stage.Stage;
```

```
la classe publique HTMLEditorSample étend l'application {
```

```
    @Passer outre
    public void start(Stage stage) {
        stage.setTitle("Composition du message");
        stage.setWidth(500); stage.setHeight(500);
        Scène scène = nouvelle scène (nouveau groupe
());
        Racine VBox finale = nouvelle VBox();
        root.setPadding(nouveaux inserts(8, 8, 8, 8));
        root.setSpacing(5); root.setAlignment(Pos.BOTTOM_LEFT);

        grille GridPane finale = new GridPane();
        grille.setVgap(5);
```

```
grille.setHgap(10);

dernier ChoiceBox sendTo =
    new ChoiceBox(FXCollections.observableArrayList(
        "Añ:", "Ccñ:", "Cciñ:")
);

sendTo.setPrefWidth(100);
GridPane.setConstraints(sendTo, 0, 0);
grille.getChildren().add(sendTo);

final TextField tbTo = new TextField(); tbTo.setPrefWidth(400);
GridPane.setConstraints(tbTo, 1, 0);

grille.getChildren().add(tbTo);

Étiquette finale subjectLabel = new Label("Sujet:");
GridPane.setConstraints(subjectLabel, 0, 1); grille.getChildren().add(subjectLabel);

final TextField tbSubject = new TextField(); tbSubject.setPrefWidth(400);
GridPane.setConstraints(tbSubject, 1, 1); grille.getChildren().add(tbSubject);

root.getChildren().add(grille);

ÉditeurHTML finalÉditeurHTML = new ÉditeurHTML();
htmlEditor.setPrefHeight(370);

root.getChildren().addAll(éditeur html, nouveau bouton("Envoyer"));

Étiquette finale htmlLabel = nouvelle étiquette();
htmlLabel.setWrapText(true);

scène.setRoot(racine);
stage.setScene(scène); spectacle();

}

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

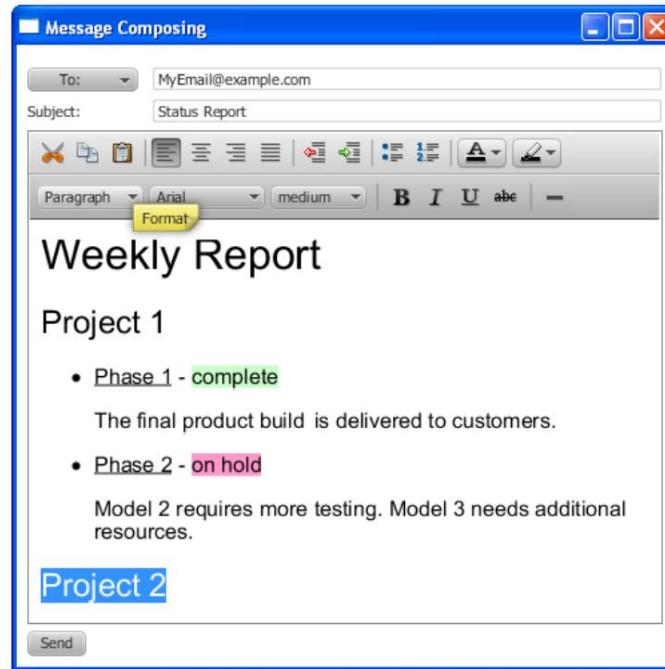
}
}
```

L'interface utilisateur comprend une boîte de choix pour sélectionner un type de destinataire, deux champs de texte pour saisir l'adresse e-mail et l'objet du message, une étiquette pour indiquer le champ d'objet, l'éditeur et le bouton Envoyer.

Les contrôles de l'interface utilisateur sont disposés sur la scène de l'application à l'aide des conteneurs de mise en page Grid et VBox. Lorsque vous compilez et exécutez cette application, la fenêtre illustrée à la [Figure 19–5](#) affiche la sortie de cette application lorsqu'un utilisateur compose un rapport hebdomadaire.

[Obtenir du contenu HTML](#)

Figure 19–5 Interface utilisateur du client de messagerie



Vous pouvez définir les valeurs de largeur et de hauteur spécifiques pour l'objet HTMLEditor en appelant les méthodes `setPrefWidth` ou `setPrefHeight`, ou vous pouvez laisser sa largeur et sa hauteur non spécifiées. [L'exemple 19–4](#) spécifie la hauteur du composant. Sa largeur est définie par le conteneur de mise en page `VBox`. Lorsque le contenu du texte dépasse la hauteur de la zone d'édition, la barre de défilement verticale apparaît.

## Obtenir du contenu HTML

Avec le contrôle JavaFX HTMLEditor, vous pouvez modifier le texte et définir le contenu initial. De plus, vous pouvez obtenir le texte saisi et modifié au format HTML. L'application illustrée dans l'[exemple 19–5](#) implémente cette tâche.

### Exemple 19–5 Récupération du code HTML

```
importer javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.stage.Stage;
```

la classe publique HTMLEditorSample étend l'application {  
 private final String INITIAL\_TEXT = "La formation est une douleur"  
 "Charmant, la zone de stockage principale.  
 + "En cours de chocolat, le canapé à l'avant. Aucune conséquence !"  
 "Les devoirs sont sélectionnés dans les membres. Aucun football n'est un  
 football à décorer On n'a pas le temps de vivre la peur de l'auteur +"ornare.

+ "Parfait pour le goût du repas.

```
@Passer outre
public void start(Stage stage) {
    stage.setTitle("HTMLEditor Sample"); stage.setWidth(500);
    stage.setHeight(500); Scène scène = nouvelle scène
    (nouveau groupe ());

    Racine VBox = nouvelle
    VBox(); root.setPadding(nouveaux inserts(8, 8, 8, 8));
    root.setSpacing(5); root.setAlignment(Pos.BOTTOM_LEFT);

    ÉditeurHTML finalÉditeurHTML = newÉditeurHTML();
    htmlEditor.setPrefHeight(245);
    htmlEditor.setHtmlText(INITIAL_TEXT);

    zone de texte finale htmlCode = nouvelle zone de texte
    (); htmlCode.setWrapText(true);

    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane"); scrollPane.setContent(htmlCode);

    scrollPane.setFitToWidth(true);
    scrollPane.setPrefHeight(180);

    Button showHTMLButton = new Button("Produire du code HTML"); root.setAlignment(Pos.CENTER);
    showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {

        @Override public void handle(ActionEvent arg0)
        { htmlCode.setText(htmlEditor.getHtmlText());
        }
    });

    root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane); scène.setRoot(racine);

    stage.setScene(scène); spectacle();

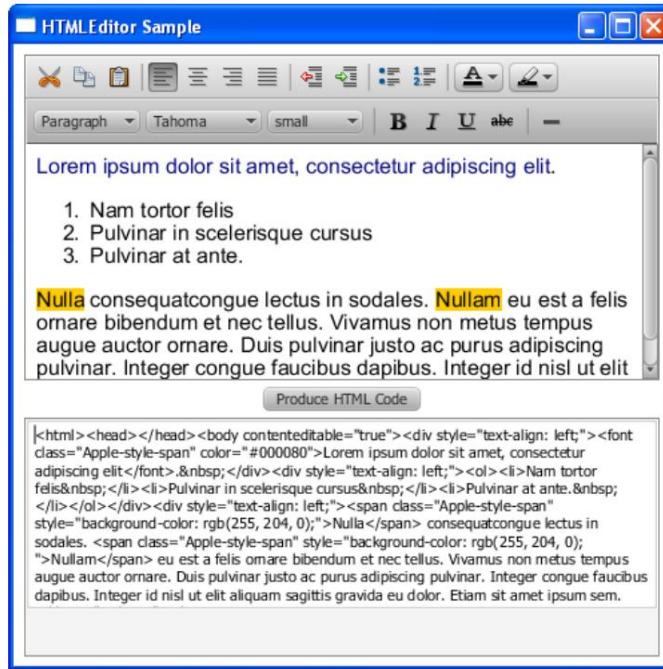
}

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

}
}
```

La méthode getHTMLText appelée sur l'objet HTMLEditor dérive le contenu modifié et le présente sous la forme d'une chaîne HTML. Ces informations sont transmises à la zone de texte, afin que vous puissiez observer, copier et coller le code HTML produit. [La Figure 19–6](#) montre un code HTML du texte en cours d'édition dans l'exemple HTMLEditor.

## Figure 19–6 Obtention du contenu HTML



De même, vous pouvez obtenir du code HTML et l'enregistrer dans le fichier ou l'envoyer à l'objet WebView pour synchroniser le contenu dans l'éditeur et le navigateur intégré. Voyez comment cette tâche est implémentée dans l'[Exemple 19–6](#).

### **Exemple 19–6 Affichage du contenu HTML modifié dans un navigateur**

```
importer javafx.application.Application; import  
javafx.event.ActionEvent; import  
javafx.event.EventHandler; importer  
javafx.geometry.Insets; importer  
javafx.geometry.Pos; importer javafx.scene.Group;  
importer javafx.scene.Scene; importer  
javafx.scene.control.*; importer  
javafx.scene.layout.VBox; importer  
javafx.scene.web.HTMLEditor; importer  
javafx.scene.web.WebEngine; importer  
javafx.scene.web.WebView; importer  
javafx.stage.Stage;
```

```
la classe publique HTMLEditorSample étend l'application {  
    private final String INITIAL_TEXT = "La formation est une douleur"  
        + "l'environnement, principale zone de stockage On n'a pas le temps de vivre la peur de  
        l'auteur +"ornare.  
        "  
  
        + "Parfait pour le goût du repas.
```

```
@Override  
public void start(Stage stage) {
```

```
stage.setTitle("HTMLEditor Sample"); stage.setWidth(500);
stage.setHeight(500); Scène scène = nouvelle scène
(nouveau groupe ());

Racine VBox = nouvelle VBox();
root.setPadding(nouveaux inserts(8, 8, 8, 8)); root.setSpacing(5);
root.setAlignment(Pos.BOTTOM_LEFT);

ÉditeurHTML final ÉditeurHTML = new ÉditeurHTML();
htmlEditor.setPrefHeight(245);
htmlEditor.setHtmlText(INITIAL_TEXT);

navigateur WebView final = new WebView(); WebEngine
final webEngine = navigateur.getEngine();

ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane"); scrollPane.setStyle("-fx-background-
color: blanc"); scrollPane.setContent(navigateur); scrollPane.setFitToWidth(true);
scrollPane.setPrefHeight(180);

Button showHTMLButton = new Button("Charger le contenu dans le navigateur");
root.setAlignment(Pos.CENTER); showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {

    @Override public void handle(ActionEvent arg0) {
        webEngine.loadContent(htmlEditor.getHtmlText());
    }
});

root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane); scène.setRoot(racine);

stage.setScene(scène); spectacle();

}

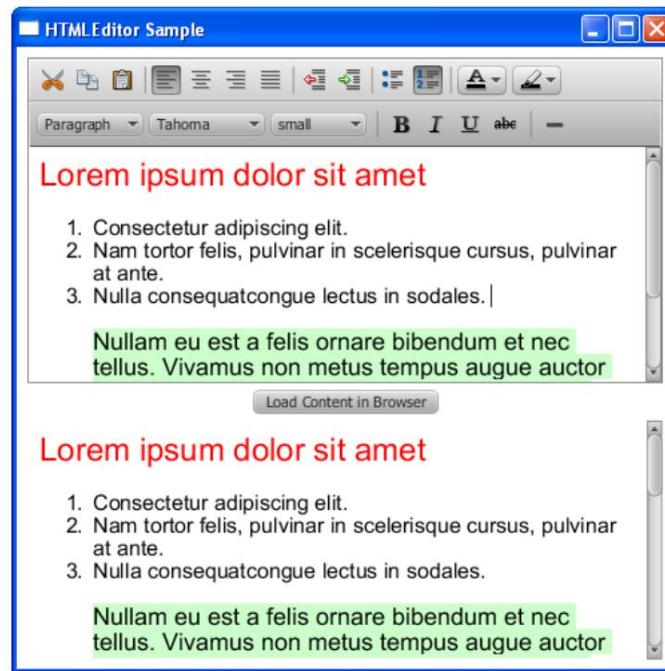
public static void main(String[] args) { launch(args);

}
}
```

Le code HTML reçu du composant htmlEditor est chargé dans l'objet WebEngine qui spécifie le contenu du navigateur intégré. Chaque fois qu'un utilisateur clique sur le bouton Charger le contenu dans le navigateur, le contenu modifié est mis à jour dans le navigateur. La figure 19-7 illustre l' [exemple 19-6](#) en action.

[Obtenir du contenu HTML](#)

Figure 19–7 Chargement de contenu dans un navigateur



Vous pouvez utiliser le composant Texte pour ajouter du contenu textuel non modifiable à votre interface utilisateur. Voir Utilisation de texte et d'effets de texte dans JavaFX pour plus d'informations sur le composant Text.

**Documentation relative à l'API**

- ÿ Éditeur HTML
- ÿ Affichage Web
- ÿ Moteur Web
- ÿ Étiquette
- ÿ Bouton
- ÿ Champ de texte
- ÿ Boîte de choix
- ÿ Panneau de défilement

# 20

## 20Info-bulle

Dans ce chapitre, vous en apprendrez plus sur l'info-bulle, le contrôle qui peut être défini pour que n'importe quel contrôle de l'interface utilisateur s'affiche lorsque ce contrôle est survolé par le curseur de la souris.

La classe Tooltip représente un composant d'interface utilisateur commun qui est généralement utilisé pour afficher des informations supplémentaires sur le contrôle d'interface utilisateur. L'info-bulle peut être définie sur n'importe quel contrôle en appelant la méthode setTooltip.

L'info-bulle a deux états différents: activé et affiché. Lorsque la bulle d'aide est activée, la souris se déplace sur un champ. Lorsque l'info-bulle est à l'état affiché, elle apparaît réellement. Une info-bulle affichée est également activée. Il y a généralement un certain délai entre le moment où l'info-bulle est activée et le moment où elle s'affiche réellement.

Un champ de mot de passe avec une info-bulle est illustré à la [Figure 20–1](#).

**Figure 20–1 Info-bulle ajoutée à un champ de mot de passe**



### Création d'une info-bulle

Étudiez le fragment de code de l' [exemple 20–1](#) qui crée le champ de mot de passe avec une info-bulle dans l'application JavaFX illustrée dans la figure précédente.

#### Exemple 20–1 Ajout d'une info-bulle au champ Mot de passe

```
champ de mot de passe final pf = nouveau champ de mot de passe();
info-bulle finale info-bulle = nouvelle info-bulle();
info-bulle.setText(
    "\nVotre mot de passe doit être\n" +
    "au moins 8 caractères de longueur\n" +
);
pf.setTooltip(info-bulle);
```

Chaque contrôle d'interface utilisateur du package javafx.scene.control a le setTooltip méthode pour ajouter une info-bulle. Vous pouvez définir une légende de texte dans un constructeur d'info-bulle ou en utilisant la méthode setText.

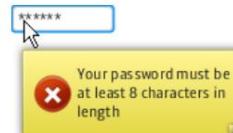
Étant donné que la classe Tooltip est une extension de la classe Labeled, vous pouvez ajouter non seulement une légende de texte, mais également une icône graphique. Le fragment de code de l' [Exemple 20–2](#) ajoute une icône à l'info-bulle du champ de mot de passe.

**Exemple 20–2 Ajout d'une icône à une info-bulle**

```
Image image = nouvelle Image(
    getClass().getResourceAsStream("warn.png")
);
tooltip.setGraphic(new ImageView(image));
```

Une fois que vous avez ajouté ce fragment de code à l'application et que le code est compilé et exécuté, l'info-bulle illustrée à la [Figure 20–2](#) s'affiche.

**Figure 20–2 Info-bulle avec une icône**



Une info-bulle peut non seulement contenir des informations supplémentaires ou auxiliaires, mais elle peut également présenter des données.

**Présentation des données d'application dans les info-bulles**

L'application de la [Figure 20–3](#) utilise les informations présentées dans les info-bulles pour calculer le coût total du séjour à l'hôtel

**Figure 20–3 Calcul des tarifs hôteliers**



Chaque case à cocher est accompagnée d'une info-bulle. Chaque info-bulle affiche le tarif d'une option de réservation particulière. Si un utilisateur coche une case, la valeur correspondante est ajoutée au total. Si une case cochée est décochée, la valeur correspondante est déduite du total.

Passez en revue le code source de l'application illustrée dans l' [Exemple 20–3](#).

**Exemple 20–3 Utilisation des info-bulles pour calculer les tarifs de l'hôtel**

```
importer javafx.application.Application;
importer javafx.beans.value.ChangeListener;
importer javafx.beans.value.ObservableValue;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.CheckBox;
importer javafx.scene.control.Label;
importer javafx.scene.control.Tooltip;
importer javafx.scene.layout.HBox;
importer javafx.scene.layout.VBox;
importer javafx.scene.text.Font;
```

```
importer javafx.stage.Stage;

la classe publique principale étend l'application {

    salles finales statiques String[] = new String[][
        "Hébergement (BB)",
        "La moitié du tableau",
        "Départ tardif",
        "Lit supplémentaire"
    ];
    taux statiques finaux Integer[] = new Integer[]{ 100, 20, 10, 30

    };
    CheckBox final[] cbs = new CheckBox[rooms.length];
    total final de
    l'étiquette = nouvelle étiquette("Total: 0$");
    Somme entière = 0;

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

    @Passer autre
    public void start(Stage stage) {
        Scene scene =
            new Scene(new Group());
        stage.setTitle("Exemple
d'info-bulle");
        stage.setWidth(300);
        stage.setHeight(150);

        total.setFont(nouvelle police("Arial", 20));

        for (int i = 0; i < rooms.length; i++) {
            final CheckBox cb =
                cbs[i] = new CheckBox(rooms[i]);
            taux entier final = taux[i];
            info-bulle finale
            tooltip = new Tooltip("$" + rate[i].toString());
            tooltip.setFont(nouvelle
            police("Arial", 16));
            cb.setTooltip(info-bulle);

            cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
                public void changé (ObservableValue<? étend Boolean> ov,
                    Boolean old_val, Boolean new_val) {
                    if
                    (cb.isSelected()) {
                        somme = somme +
                        taux;
                    } autre {
                        somme = somme - taux;
                    }
                    total.setText("Total : $" + sum.toString());
                }
            });
        }
    }

    VBox vbox = nouvelle VBox();
    vbox.getChildren().addAll(cbs);
    vbox.setSpacing(5);
    Racine HBox = nouvelle
    HBox();
    root.getChildren().add(vbox);
    root.getChildren().add(total);
    root.setSpacing(40);
    root.setPadding(nouveaux
    inserts(20, 10, 10, 20));

    ((Groupe) scene.getRoot()).getChildren().add(root);
}
```

```
        stage.setScene(scene);
        spectacle();
    }
}
```

La ligne de code de l' [Exemple 20–4](#) a été utilisée dans l' [Exemple 20–3](#) pour créer une info-bulle et lui attribuer une légende de texte. La valeur entière du prix de l'option a été convertie en valeur de chaîne.

#### **Exemple 20–4 Définition de la valeur d'une info-bulle**

info-bulle finale tooltip = new Tooltip("\$" + rate[i].toString())

Vous pouvez modifier l'apparence visuelle d'une info-bulle en appliquant CSS.

#### **Documentation relative à l'API**

ÿ Info-bulle

ÿ Étiqueté

# 21

## 21 Volet titré et accordéon

Ce chapitre explique comment utiliser une combinaison des volets accordéon et titre dans vos applications JavaFX.

Un volet intitulé est un panneau avec un titre. Il peut être ouvert et fermé, et il peut encapsuler n'importe quel nœud tel que des contrôles ou des images d'interface utilisateur, et des groupes d'éléments ajoutés à un conteneur de mise en page.

Les volets titrés peuvent être regroupés à l'aide de la commande accordéon, qui vous permet de créer plusieurs volets et de les afficher un à la fois. [La Figure 21-1](#) montre une commande en accordéon qui combine trois volets titrés.

**Figure 21-1 Accordéon avec trois volets titrés**



Utilisez les classes Accordion et TitledPane dans l'API JavaFX SDK pour implémenter ces contrôles dans vos applications.

### Création de volets titrés

Pour créer un contrôle TitledPane, définissez un titre et du contenu. Vous pouvez le faire en utilisant le constructeur à deux paramètres de la classe TitledPane ou en appliquant les méthodes setText et setContent. Les deux méthodes sont illustrées dans [l'exemple 21-1](#).

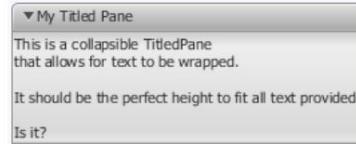
#### Exemple 21-1 Déclaration d'un objet TitledPane

```
// en utilisant un constructeur à deux paramètres
TitledPane tp = new TitledPane("Mon volet titré", new Button("Button"));
//appliquer des méthodes
TitledPane tp = new TitledPane();
tp.setText("Mon volet titré");
tp.setContent(nouveau bouton("Bouton"));
```

La compilation et l'exécution d'une application avec l'un ou l'autre des fragments de code produit le contrôle illustré à la [Figure 21-2](#).

**Figure 21–2 Volet intitulé avec un bouton**

Le volet intitulé est redimensionné pour s'adapter à la taille préférée de son contenu. Vous pouvez ajouter du texte multiligne et évaluer le résultat comme illustré à la [Figure 21–3](#).

**Figure 21–3 Volet intitulé avec du texte**

Ne définissez pas explicitement la hauteur minimale, maximale ou préférée d'un volet intitulé, car cela peut entraîner un comportement inattendu lorsque le volet intitulé est ouvert ou fermé.

Le fragment de code illustré dans l' [Exemple 21–2](#) ajoute plusieurs contrôles au volet intitulé en les plaçant dans le conteneur de disposition GridPane.

#### **Exemple 21–2 Volet intitulé avec le conteneur de disposition GridPane**

```
TitledPane gridTitlePane = new TitledPane();
GridPane grid = new GridPane();
grille.setVgap(4);
grid.setPadding(nouveaux inserts(5, 5, 5, 5));
grid.add(new Label("Prénom : "), 0, 0);
grille.add(nouveau TextField(), 1, 0);
grid.add(new Label("Nom : "), 0, 1);
grille.add(nouveau TextField(), 1, 1);
grid.add(new Label("Email: "), 0, 2);
grille.add(nouveau TextField(), 1, 2);
gridTitlePane.setText("Grille");
gridTitlePane.setContent(grid);
```

Lorsque vous exécutez et compilez une application avec ce fragment de code, la sortie illustrée à la [Figure 21–4](#) s'affiche.

**Figure 21–4 Volet intitulé contenant plusieurs contrôles**

Vous pouvez définir la façon dont un volet intitulé s'ouvre et se ferme. Par défaut, tous les volets intitulés sont réductibles et leurs mouvements sont animés. Si votre application interdit la fermeture d'un volet intitulé, utilisez la méthode setCollapsible avec la valeur false. Vous pouvez également désactiver l'ouverture animée en appliquant la méthode setAnimated avec la valeur false évaluer. Le fragment de code présenté dans l' [Exemple 21–3](#) implémente ces tâches.

### Exemple 21–3 Réglage du style d'un volet titré

```
TitledPane tp = new TitledPane(); //interdit la
fermeture de tp.setCollapsible(false); //interdit
d'animer tp.setAnimated(false);
```

#### Ajout de volets titrés à un accordéon Dans vos

applications, vous pouvez utiliser des volets titrés en tant qu'éléments autonomes ou vous pouvez les combiner dans un groupe à l'aide de la commande Accordion. Ne définissez pas explicitement la hauteur minimale, maximale ou préférée d'un accordéon, car cela peut entraîner un comportement inattendu lorsque l'un de ses volets intitulés est ouvert.

L'ajout de plusieurs volets titrés à un accordéon est similaire à l'ajout de boutons bascule à un groupe bascule : un seul volet intitulé peut être ouvert dans un accordéon à la fois.

[L'exemple 21–4](#) crée trois volets intitulés et les ajoute à un accordéon.

### Exemple 21–4 Accordéon et trois volets titrés

```
importer javafx.application.Application;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Accordion;
importer javafx.scene.control.TitledPane;
importer javafx.scene.image.Image;
importer javafx.scene.image.ImageView;
importer javafx.scene.paint.Color;
importer javafx.stage.Stage;
```

```
la classe publique TitledPaneSample étend l'application {
    final String[] imageNames = new String[]{"Pommes", "Fleurs", "Feuilles"};
    Image finale [] images =
        nouvelle image [imageNames.length];
    ImageView final [] pics = new ImageView[imageNames.length];
    final TitledPane[] tps = new TitledPane[imageNames.length];
```

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

@Override public void start(Stage stage)
{ stage.setTitle("TitledPane");
Scene scene = new
Scene(new Group(), 80, 180);
scene.setFill(Color.GHOSTWHITE);
```

**Accordéon final accordéon = nouvel Accordéon ();**

```
for (int i = 0; i < imageNames.length; i++) { images[i] = new
Image(getClass().getResourceAsStream(imageNames[i]
+ ".jpg"));
pics[i] = new ImageView(images[i]);
tps[i] = new
TitledPane(imageNames[i],pics[i]);
}
```

```
accordéon.getPanels().addAll(tps);
accordéon.setExpandedPanel(tps[0]);
```

Racine de groupe = (Groupe)scene.getRoot();

```

        root.getChildren().add(accordéon);
        stage.setScene(scene);
        spectacle();
    }
}

```

Trois volets intitulés sont créés dans la boucle. Le contenu de chaque volet intitulé est défini en tant qu'objet ImageView. Les volets intitulés sont ajoutés à l'accordéon à l'aide des méthodes getPanels et addAll. Vous pouvez utiliser la méthode add au lieu de addAll méthode pour ajouter un seul volet intitulé.

Par défaut, tous les volets titrés sont fermés au démarrage de l'application. La méthode setExpandedPane de l' [exemple 21–4](#) spécifie que le volet intitulé avec l'image Pommes s'ouvrira lorsque vous exécuterez l'exemple, comme illustré à la [figure 21–5](#).

**Figure 21–5 Accordéon avec trois volets titrés**



## Traitement des événements pour un accordéon avec des volets titrés

Vous pouvez utiliser des volets titrés et des accordéons pour présenter différentes données dans vos applications. L'[exemple 21–5](#) crée un volet intitulé autonome avec le conteneur de disposition GridPane et trois volets intitulés combinés à l'aide de l'accordéon. Le volet intitulé autonome contient des éléments d'interface utilisateur d'un client de messagerie. L'accordéon permet à la sélection d'une image d'apparaître dans le champ Pièce jointe du volet intitulé Grille.

### Exemple 21–5 Implémentation de ChangeListener pour un accordéon

```

importer javafx.application.Application;
importer javafx.beans.value.ChangeListener;
importez javafx.beans.value.ObservableValue;
importer javafx.geometry.Insets;
importer javafx.scene.Group;
importer javafx.scene.Scene;
importer javafx.scene.control.Accordion;
importer javafx.scene.control.Label;
importer javafx.scene.control.TextField;
importer javafx.scene.control.TitledPane;
importer javafx.scene.image.ImageView;
importer javafx.scene.image.Image;
importer javafx.scene.layout.GridPane;
importer javafx.scene.layout.HBox;
importer javafx.scene.paint.Color;
importer javafx.stage.Stage;

la classe publique TitledPaneSample étend l'application {
    final String[] imageNames = new String[]{"Pommes", "Fleurs", "Feuilles"};
    Image finale [] images = nouvelle image [imageNames.length];
    ImageView final[] pics = new ImageView[imageNames.length];
}

```

```

final TitledPane[] tps = new TitledPane[imageNames.length]; Étiquette finale étiquette =
nouvelle étiquette("N/A");

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

}

@Override public void start(Stage stage) { stage.setTitle("TitledPane");
Scene scene = new Scene(new Group(), 800, 250);
scene.setFill(Color.GHOSTWHITE);

// --- Conteneur GridPane
TitledPane gridTitlePane = new TitledPane(); GridPane grid = new
GridPane(); grille.setVgap(4); grid.setPadding(nouveaux inserts(5, 5, 5,
5)); grid.add(new Label("Toÿ: "), 0, 0); grille.add(nouveau TextField(), 1,
0); grid.add(new Label("Cc: "), 0, 1); grille.add(nouveau TextField(), 1,
1); grid.add(new Label("Objet : "), 0, 2); grille.add(nouveau TextField(),
1, 2); grid.add(new Label("Pièce jointe : "), 0, 3); grille.add(étiquette, 1,
3); gridTitlePane.setText("Grille"); gridTitlePane.setContent(grid);

// --- Accordéon
Accordéon final accordéon = nouvel Accordéon (); for (int i = 0; i <
imageNames.length; i++) { images[i] = new
Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
pics[i] = new ImageView(images[i]); tps[i] = new TitledPane(imageNames[i],pics[i]);

}

accordéon.getPanes().addAll(tps);
accordion.expandedPaneProperty().addListener(new
ChangeListener<TitledPane>() {
    public void changé (ObservableValue<? étend TitledPane> ov, TitledPane old_val, TitledPane
new_val) { if (new_val != null) {

        label.setText(accordion.getExpandedPane().getText() +
        ".jpg");
    }
});
};

HBox hbox = nouvelle HBox(10);
hbox.setPadding(nouveaux inserts(20, 0, 0, 20));
hbox.getChildren().addAll(gridTitlePane, accordéon);

Racine de groupe = (Groupe)scene.getRoot();
root.getChildren().add(hbox); stage.setScene(scene);
spectacle();

}
}
}

```

Lorsqu'un utilisateur ouvre un volet intitulé dans l'accordéon, la propriété `extendedPaneProperty` de l'accordéon change. L'objet `ChangeListener` est informé de cette modification et le volet intitulé développé dans l'accordéon est utilisé pour construire un nom de fichier de la pièce jointe. Ce nom de fichier est défini comme texte de l'objet `Label` correspondant.

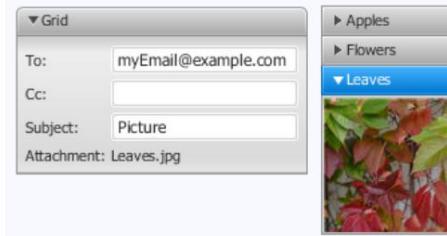
La [Figure 21–6](#) montre à quoi ressemble l'application après son démarrage. L'étiquette de pièce jointe contient "N/A", car aucun des volets intitulés n'est sélectionné.

**Figure 21–6 Vue initiale de l'application TitledPaneSample**



Si vous développez le volet intitulé Feuilles, l'étiquette de la pièce jointe contiendra "Leaves.jpg", comme illustré à la [Figure 21–7](#).

**Figure 21–7 Exemple d'application TitledPane avec le volet intitulé Leaves développé**



Étant donné que les classes `TitledPane` et `Accordion` sont toutes deux des extensions de `Node` classe, vous pouvez leur appliquer des effets visuels ou des transformations. Vous pouvez également modifier l'apparence des contrôles en appliquant des styles CSS.

#### Documentation relative à l'API

- ÿ Volet titré
- ÿ Accordéon
- ÿ Étiquette
- ÿ Volet Grille
- ÿ Champ de texte

# 22

## 22Menu

Ce chapitre explique comment créer des menus et des barres de menus, ajouter des éléments de menu, regrouper les menus en catégories, créer des sous-menus et définir des menus contextuels.

Vous pouvez utiliser les classes suivantes de l'API JavaFX pour créer des menus dans votre application JavaFX.

Barre de menus \_

ÿ Élément de menu

- Menus

- CheckMenuItem

- RadioMenuItem

- Menus

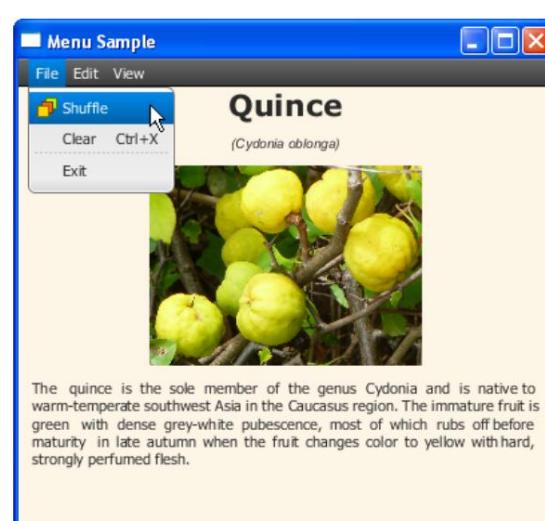
- Élément de menu personnalisé

- \* élément de menu séparateur

ÿ Menu contextuel

[La Figure 22–1 montre une capture d'écran d'une application avec une barre de menus typique.](#)

**Figure 22–1 Application avec une barre de menus et trois catégories de menus**



## Création de menus dans les applications JavaFX

Un menu est une liste d'éléments exploitables qui peuvent être affichés à la demande d'un utilisateur. Lorsqu'un menu est visible, les utilisateurs peuvent sélectionner un élément de menu à la fois. Lorsqu'un utilisateur clique sur un élément, le menu revient en mode masqué. En utilisant les menus, vous pouvez économiser de l'espace dans l'interface utilisateur (UI) de votre application en plaçant dans les menus les fonctionnalités qui n'ont pas toujours besoin d'être visibles.

Les menus d'une barre de menus sont généralement regroupés en catégories. Le modèle de codage consiste à déclarer une barre de menus, à définir les menus de catégorie et à remplir les menus de catégorie avec des éléments de menu. Utilisez les classes d'éléments de menu suivantes lors de la création de menus dans vos applications JavaFX:

- ÿ MenuItem - pour créer une option actionnable
- ÿ Menu – pour créer un sous-menu
- ÿ RadioButtonItem – pour créer une sélection mutuellement exclusive
- ÿ CheckMenuItem - pour créer une option qui peut être basculée entre les états sélectionnés et non sélectionnés

Pour séparer les éléments de menu au sein d'une même catégorie, utilisez la classe SeparatorMenuItem.

Les menus organisés par catégories dans une barre de menus sont généralement situés en haut de la fenêtre, laissant le reste de la scène pour les éléments cruciaux de l'interface utilisateur. Si, pour certaines raisons, vous ne pouvez attribuer aucune partie visuelle de votre interface utilisateur à une barre de menus, vous pouvez utiliser des menus contextuels que l'utilisateur ouvre d'un clic de souris.

## Création d'une barre de menus

Bien qu'une barre de menus puisse être placée ailleurs dans l'interface utilisateur, elle est généralement située en haut de l'interface utilisateur et contient un ou plusieurs menus. La barre de menus se redimensionne automatiquement pour s'adapter à la largeur de la fenêtre de l'application. Par défaut, chaque menu ajouté à la barre de menus est représenté par un bouton avec la valeur textuelle.

Considérez une application qui affiche des informations de référence sur les plantes telles que leur nom, leur nom binomial, une image et une brève description. Vous pouvez créer trois catégories de menu: Fichier, Modifier et Afficher, et les remplir avec les éléments de menu.

L'[exemple 22-1](#) montre le code source d'une telle application avec la barre de menus ajoutée.

### Exemple 22-1 Exemple d'application de menu

```
importer java.util.AbstractMap.SimpleEntry;
importer java.util.Map.Entry;
importer javafx.application.Application;
importer javafx.scene.Scene;
importer javafx.scene.control.*;
importer javafx.scene.effect.DropShadow;
importer javafx.scene.effect.Effect;
importer javafx.scene.effect.Glow;
importer javafx.scene.effect.SepiaTone;
importer javafx.scene.image.Image;
importer javafx.scene.image.ImageView;
importer javafx.scene.layout.VBox;
importer javafx.scene.paint.Color;
importer javafx.stage.Stage;
```

```
la classe publique MenuSample étend l'application {
```

```

PageData final [] pages = new PageData [] {
    new PageData("Pomme", "La
        pomme est le fruit à pépins du pommier, espèce Malus " + "domestica de la famille des roses (Rosaceae). C'est "
        l'un des + "fruits d'arbre les plus cultivés, et le plus largement connu "+" des nombreux membres du genre Malus
        qui sont utilisés par l'homme. "
        +
        + "L'arbre est originaire d'Asie occidentale, où son ancêtre sauvage, + "l'Alma, se trouve encore aujourd'hui.", "
        "Malus domestica"),
    new PageData("Aubépine",
        « L'aubépine est un grand genre d'arbustes et d'arbres de la famille des rosiers +, les Rosacées,
        originaire des régions tempérées du Nord »
        +
        + " Hémisphère en Europe, Asie et Amérique du Nord. "
        +
        + Le nom aubépine était
        +
        + "appliqué à l'origine à l'espèce originaire du nord de l'Europe, " + "en particulier l'aubépine commune C.
        monogyna, et le " + "nom non modifié est souvent ainsi utilisé en Grande-Bretagne et en Irlande.", "Crataegus
        monogyna"), new PageData( "Lierre", "Le lierre est une plante à fleurs de la famille des raisins (Vitacées)
        originaire d'Asie orientale au Japon, en Corée, et dans le nord et l'est de la Chine."
        +
        +
        + "Il s'agit d'une liane ligneuse à feuilles caduques atteignant 30 m de haut ou plus sur " + "un support approprié,
        se fixant au moyen de nombreuses " + "petites vrilles ramifiées terminées par des disques collants.", "Parthenocissus
        tricuspidata"),
    new PageData("Coing", "Le coing
        est le seul membre du genre Cydonia et est originaire de + "l'Asie du sud-ouest tempérée chaude dans la région
        du Caucase. Le " + "fruit immature est vert avec une pubescence gris-blanc dense, la plupart + "dont déteint avant
        maturité à la fin de l'automne lorsque le fruit " + "change de couleur en jaune avec une chair dure et fortement "
        parfumée.", "Cydonia oblonga")
}

};

chaîne finale [] options d'affichage = nouvelle chaîne [] {
    "Titre",
    "Nom binomial",
    "Image",
    "La description"
};

entrée finale<chaîne, effet>[] effets = nouvelle entrée[]
    new SimpleEntry<String, Effect>("Sepia Tone", new SepiaTone()), new SimpleEntry<String, Effect>"Glow",
    new Glow()), new SimpleEntry<String, Effect>("Shadow", new Ombre portée())

};

pic ImageView final = new ImageView(); nom final de l'étiquette
= new Label();
Etiquette finale binName = new Label(); description finale
de l'étiquette = nouvelle étiquette ();

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

}

@Passer outre
public void start(Stage stage) {
    stage.setTitle ("Échantillon de menu");
}

```

```

Scene scene = new Scene(new VBox(), 400, 350);
scene.setFill(Color.OLDLACE);

MenuBar menuBar = newMenuBar();

// --- Menu Fichier
Menu menuFichier = new Menu("Fichier");

// --- Menu Edition
Menu menuEdit = new Menu("Edit");

// --- Affichage des menus
Menu menuAffichage = new Menu("Affichage");

menuBar.getMenus().addAll(menuFile, menuEdit, menuView);

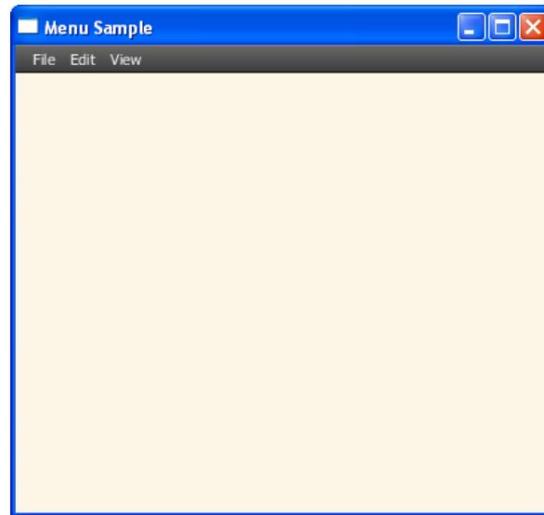
((VBox) scene.getRoot()).getChildren().addAll(menuBar);

stage.setScene(scene);
spectacle();
}

classe privée PageData { nom de
la chaîne publique;
description de la chaîne publique;
public String binNames; image
publique image; public PageData(String
name, String description, String binNames) { this.name = name; this.description = description;
this.binNames = binNames; image = nouvelle Image(getClass().getResourceAsStream(nom +
".jpg"));

}
}
}

Contrairement aux autres contrôles d'interface utilisateur, la classe Menu et les autres extensions de la
classe MenuItem n'étendent pas la classe Node. Ils ne peuvent pas être ajoutés directement à la scène de
l'application et restent invisibles jusqu'à ce qu'ils soient ajoutés à la barre de menus via la méthode getMenus.
```

**Figure 22–2 La barre de menus est ajoutée à l'application**

Vous pouvez naviguer dans les menus en utilisant les touches fléchées du clavier. Cependant, lorsque vous sélectionnez un menu, aucune action n'est effectuée, car le comportement des menus n'est pas encore défini.

## Ajout d'éléments de menu

Définissez la fonctionnalité du menu Fichier en ajoutant les éléments suivants :   
    ÿ Lecture aléatoire – pour charger des informations de référence sur les plantes  
    ÿ Effacer – pour supprimer les informations de référence et effacer la scène  
    ÿ Separator – pour détacher des éléments de menu  
    ÿ Exit – pour quitter l'application Les lignes en gras dans l'[Exemple 22–2](#) créent un menu Shuffle à l'aide de la classe MenuItem et ajoutent des composants graphiques à la scène de l'application. La classe MenuItem permet de créer un élément exploitable avec du texte et des graphiques. L'action effectuée sur un clic de l'utilisateur est définie par la méthode setOnAction, similaire à la classe Button.

### Exemple 22–2 Ajout de l'élément de menu aléatoire avec des graphiques

```
importer java.util.AbstractMap.SimpleEntry; import
java.util.Map.Entry; import javafx.application.Application;
import javafx.event.ActionEvent; import
javafx.event.EventHandler; import
javafx.geometry.Insets; import javafx.geometry.Pos;
import javafx.scene.Scene; import
javafx.scene.control.*; import
javafx.scene.effect.DropShadow; import
javafx.scene.effect.Effect; import
javafx.scene.effect.Glow; import
javafx.scene.effect.SepiaTone; import
javafx.scene.image.Image; import
javafx.scene.image.ImageView; import
javafx.scene.layout.VBox; import
javafx.scene.paint.Color;
```

Ajout d'éléments de menu

```

importer javafx.scene.text.Font; importer
javafx.scene.text.TextAlignment; importer javafx.stage.Stage;

la classe publique MenuSample étend l'application {

    PageData final [] pages = new PageData [] {
        new PageData("Pomme", "La
            pomme est le fruit à pépins du pommier, espèce Malus +" + "domestica de la famille des roses (Rosaceae). C'est
            l'un des + "fruits d'arbre les plus cultivés, et le plus largement connu " + " des nombreux membres du genre Malus
            qui sont utilisés par l'homme. "
            +"L'arbre est originaire d'Asie occidentale, où son ancêtre sauvage, +" + "l'Alma, se trouve encore aujourd'hui.", "
            "Malus domestica"), new PageData("Aubépine",
            "« L'aubépine est un grand genre d'arbustes et d'arbres de la famille des rosiers +, les Rosacées,
            originaire des régions tempérées du Nord »
            + " Hémisphère en Europe, Asie et Amérique du Nord. "
            + "Le nom d'aubépine était
            + "appliqué à l'origine à l'espèce originaire du nord de l'Europe, " + "en particulier l'aubépine commune C.
            monogyna, et le nom " + "non modifié est souvent ainsi utilisé en Grande-Bretagne et en Irlande.", "
            "Crataegus monogyna"), new
        PageData("Ivy", "Le lierre est une plante à
            fleurs de la famille des raisins (Vitaceae) originaire" + " de l'Asie orientale au Japon, en Corée et dans le nord et
            l'est de la Chine."
            + " C'est une liane ligneuse à feuilles caduques atteignant 30 m de haut ou plus sur " + "un support approprié, se
            fixant au moyen de nombreuses petites " + "villes ramifiées terminées par des disques collants.", "Parthenocissus
            tricuspidata"),
        new PageData("Coing",
            "Le coing est le seul membre du genre Cydonia et est originaire" + " de l'Asie du sud-ouest tempérée chaude
            dans la région du Caucase. Le " + "fruit immature est vert avec une pubescence gris-blanc dense, dont la plupart
            + "se détache avant maturité en fin d'automne lorsque le fruit " + "change de couleur en jaune avec une chair
            dure et fortement parfumée.", "Cydonia oblonga")
    }

    chaîne finale [] options d'affichage = nouvelle chaîne [] {
        "Titre",
        "Nom binomial",
        "Image",
        "La description"
    }

    entrée finale<chaîne, effet>[] effets = nouvelle entrée[] {
        new SimpleEntry<String, Effect>("Sepia Tone", new SepiaTone()), new SimpleEntry<String, Effect>("Glow",
        new Glow(), new SimpleEntry<String, Effect>("Shadow", new Ombre portée()))
    }

    pic ImageView final = new ImageView(); nom final de l'étiquette
    = new Label(); Etiquette finale binName = new Label();
    description finale de l'étiquette = nouvelle étiquette (); privé int
    indexactuel = -1;
}

public static void main(String[] args) {

```

```

lancement(arguments);
}

@Passer outre
public void start(Stage stage) {
    stage.setTitle("Échantillon de menu"); Scene
    scene = new Scene(new VBox(), 400, 350);
    scene.setFill(Color.OLDLACE);

    name.setFont(new Font("Verdana Bold", 22));
    binName.setFont(nouvelle police("Arial Italic", 10));
    pic.setFitHeight(150);
    pic.setPreserveRatio(true);
    description.setWrapText(true);
    description.setTextAlignment(TextAlignment.JUSTIFY);

    mélanger();

    MenuBar menuBar = new MenuBar();

    VBox finale vbox = nouvelle VBox();
    vbox.setAlignment(Pos.CENTRE);
    vbox.setSpacing(10); vbox.setPadding(nouveaux
    inserts(0, 10, 0, 10)); vbox.getChildren().addAll(name, binName, pic,
    description);

    // --- Menu Fichier
    Menu menuFichier = new Menu("Fichier");
    MenuItem add = new MenuItem("Shuffle",
        new ImageView(new Image("menusample/new.png")));
    add.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent t) { shuffle();
            vbox.setVisible(true);

        }
    });
    menuFile.getItems().addAll (ajouter);

    // --- Menu Edition
    Menu menuEdit = new Menu("Edit");

    // --- Affichage des menus
    Menu menuAffichage = new Menu("Affichage");

    menuBar.getMenus().addAll(menuFile, menuEdit, menuView); ((VBox)
    scene.getRoot().getChildren().addAll(menuBar, vbox); stage.setScene(scene); spectacle());

}

mélange vide privé () {
    int je = indexcourant; tandis
    que (i == indexcourant) {
        i = (int) (Math.random() * pages.length);
    }
    pic.setImage(pages[i].image);
    nom.setText(pages[i].nom);
    binName.setText("(" + pages[i].binNames + ")");
    description.setText(pages[i].description);
}

```

```

        indexcourant = ijj;
    }

    classe privée PageData {
        nom de chaîne public; y;
        description de la chaîne publique; y;
        public String binNames; y;
        image publique image; y;
        public PageData(String name, String description, String binNames) {
            this.name = nom;
            this.description = description;
            this.binNames = binNames;
            image = nouvelle Image(getClass().getResourceAsStream(nom + ".jpg"));
        }
    }
}

```

Lorsqu'un utilisateur sélectionne l'élément de menu Shuffle, la méthode shuffle appelée dans setOnAction spécifie le titre, le nom binomial, une image de la plante et sa description en calculant l'index des éléments dans les tableaux correspondants.

L'élément de menu Effacer permet d'effacer la scène de l'application. Vous pouvez implémenter cela en rendant invisible le conteneur VBox avec les éléments de l'interface graphique, comme illustré dans l'[exemple 22–3](#).

#### **Exemple 22–3 Création de l'élément de menu Clear avec Accelerator**

```

MenuItem clear = new MenuItem("Effacer");
clear.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
clear.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent t) {
        vbox.setVisible(faux);
    }
});

```

L'implémentation de la classe MenuItem permet aux développeurs de définir un accélérateur de menu, une combinaison de touches qui exécute la même action que l'élément de menu. Avec le menu Effacer, les utilisateurs peuvent soit sélectionner l'action dans la catégorie du menu Fichier, soit appuyer simultanément sur la touche Contrôle et la touche X.

Le menu Quitter ferme la fenêtre de l'application. Définissez System.exit(0) comme action pour cet élément de menu, comme illustré dans l'[Exemple 22–4](#).

#### **Exemple 22–4 Création de l'élément de menu Quitter**

```

MenuItem exit = new MenuItem("Quitter");
exit.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent t) {
        System.exit(0);
    }
});

```

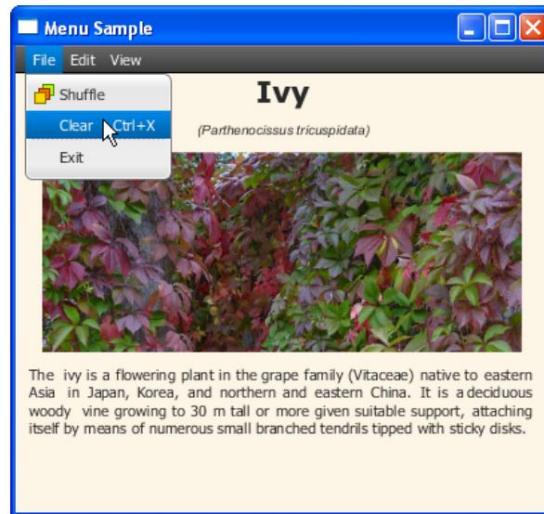
Utilisez la méthode getItems illustrée dans [l'Exemple 22–5](#) pour ajouter les éléments de menu nouvellement créés au menu Fichier. Vous pouvez créer un élément de menu séparateur et l'ajouter dans la méthode getItems pour détacher visuellement l'élément de menu Quitter.

#### **Exemple 22–5 Ajout d'éléments de menu**

```
menuFile.getItems().addAll(ajouter, effacer, nouveau SeparatorMenuItem(), exit);
```

Ajoutez l'[exemple 22-2](#), l'[exemple 22-3](#), l'[exemple 22-4](#) et l'[exemple 22-5](#) à l'application Menu Sample, puis compilez-la et exécutez-la. Sélectionnez l'élément de menu Shuffle pour charger des informations de référence sur différentes plantes. Effacez ensuite la scène (Effacer) et fermez l'application (Quitter). La [Figure 22-3](#) montre la sélection de l'élément de menu Effacer.

**Figure 22-3 Menu Fichier avec trois éléments de menu**



Avec le menu Affichage, vous pouvez masquer et afficher des éléments d'informations de référence. Implémentez la méthode `createMenuItem` et appelez-la dans la méthode `start` pour créer quatre objets `CheckMenuItem`. Ajoutez ensuite les éléments de menu de contrôle nouvellement créés au menu Affichage pour basculer la visibilité du titre, du nom binomial, de l'image de la plante et de sa description. L'[exemple 22-6](#) montre deux fragments de code qui implémentent ces tâches.

#### Exemple 22-6 Application de la classe CheckMenuItem pour créer des options de basculement

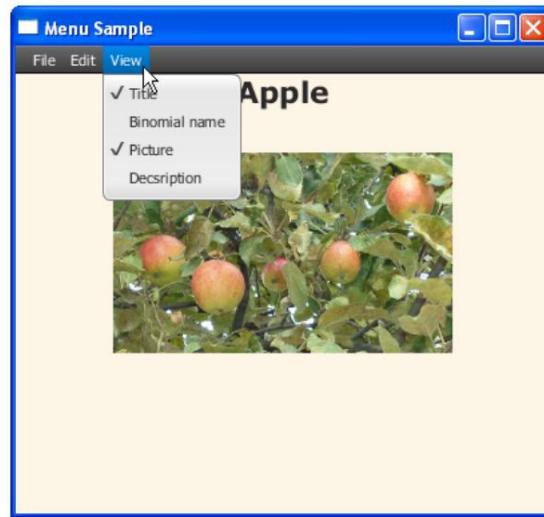
```
// --- Création de quatre éléments de menu de contrôle dans la méthode de démarrage
CheckMenuItem titleView = createMenuItem ("Titre", nom); CheckMenuItem
binNameView = createMenuItem ("Nom binomial", binName); CheckMenuItem picView = createMenuItem
("Image", image); CheckMenuItem descriptionView = createMenuItem ("Description", description);
menuView.getItems().addAll(titleView, binNameView, picView, descriptionView);

...
// La méthode createMenuItem
private static CheckMenuItem createMenuItem (titre de la chaîne, nœud final du nœud) {
    CheckMenuItem cmi = new CheckMenuItem(titre);
    cmi.setSelected(true);
    cmi.selectedProperty().addListener(new ChangeListener<Boolean>() {
        public void modifié (ObservableValue ov,
        Ancienne_val booléenne, nouvelle_val booléenne) {
            node.setVisible(new_val);
        }
    });
    retour cmi;
}
```

La classe `CheckMenuItem` est une extension de la classe `MenuItem`. Il peut être basculé entre les états sélectionnés et désélectionnés. Lorsqu'il est sélectionné, un élément de menu de vérification affiche une coche.

L' [exemple 22-6](#) crée quatre objets `CheckMenuItem` et traite la modification de leur propriété `selectedProperty`. Lorsque, par exemple, un utilisateur désélectionne le `picView` item, la méthode `setVisible` reçoit la valeur `false`, l'image de la plante devient invisible. Lorsque vous ajoutez ce fragment de code à l'application, compilez et exécutez l'application, vous pouvez tester la sélection et la désélection des éléments de menu. La [Figure 22-4](#) montre l'application au moment où le titre et l'image de la plante sont affichés, mais son nom binomial et sa description sont masqués.

Figure 22-4 Utilisation des éléments du menu Vérifier



## Création de sous-menus

Pour le menu Edition, définissez deux éléments de menu : Effet d'image et Aucun effet. L'élément de menu Effet d'image est conçu comme un sous-menu avec trois éléments pour définir l'un des trois effets visuels disponibles. L'élément de menu Aucun effet supprime l'effet sélectionné et restaure l'état initial de l'image.

Utilisez la classe `RadioMenuItem` pour créer les éléments du sous-menu. Ajoutez les boutons de menu radio à un groupe de basculement pour rendre la sélection mutuellement exclusive. L'[exemple 22-7](#) implémente ces tâches.

**Exemple 22-7** Création d'un sous-menu avec des éléments de menu radio

```
//Menu Effet d'image
Menu menuEffect = new Menu("Effet d'image");
final ToggleGroup groupEffect = new ToggleGroup();
for (Entry<String, Effect> effect : effects) {
    RadioMenuItem itemEffect = new RadioMenuItem(effect.getKey());
    itemEffect.setUserData(effect.getValue());
    itemEffect.setToggleGroup(groupEffect);
    menuEffect.getItems().add(itemEffect);
}
//Menu sans effets
final MenuItem noEffects = new MenuItem("Aucun effet");
```

```

noEffects.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent t) { pic.setEffect(null);
        groupEffect.getSelectedToggle().setSelected(false);

    }
});

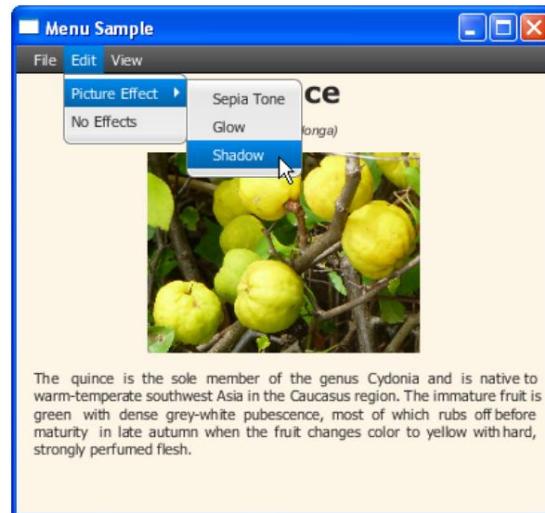
//Traitement de la sélection de l'élément de
menu groupEffect.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
    public void changé (ObservableValue<? étend Toggle> ov, Toggle old_toggle,
        Toggle new_toggle) { if (groupEffect.getSelectedToggle() != null) { Effect effect
            =
            (Effet) groupEffect.getSelectedToggle().getUserData();
            pic.setEffect(effect);
        }
    }
});
//Ajout d'éléments au menu Edition
menuEdit.getItems().addAll(menuEffect, noEffects);

```

La méthode `setUserData` définit un effet visuel pour un élément de menu radio particulier. Lorsqu'un des éléments du groupe bascule est sélectionné, l'effet correspondant est appliqué à l'image. Lorsque l'élément de menu `Aucun effet` est sélectionné, la méthode `setEffect` spécifie la valeur nulle et aucun effet n'est appliqué à l'image.

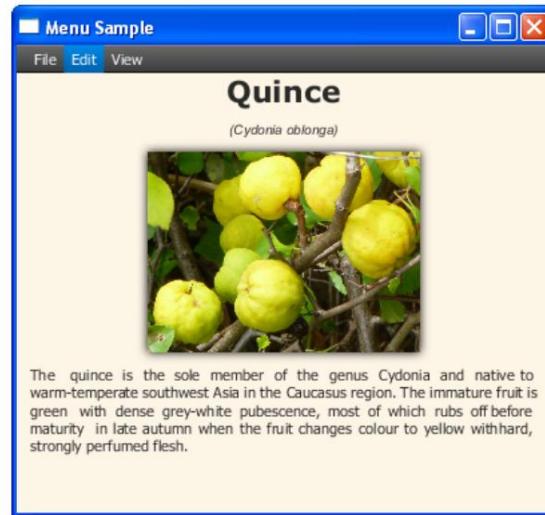
[La Figure 22–5](#) capture un moment où un utilisateur sélectionne un élément du menu `Shadow`.

**Figure 22–5 Sous-menu avec trois éléments de menu radio**



Lorsque l'effet `DropShadow` est appliqué à l'image, elle ressemble à celle illustrée à la [Figure 22–6](#).

Figure 22–6 Image de coing avec un effet DropShadow appliqué



Vous pouvez utiliser la méthode `setDisable` de la classe `MenuItem` pour désactiver le menu Aucun effet lorsqu'aucun des effets n'est sélectionné dans le sous-menu Effet d'image. Modifiez l'[Exemple 22–7](#) comme indiqué dans l'[Exemple 22–8](#).

#### Exemple 22–8 Désactivation d'un élément de menu

```
Menu menuEffect = new Menu("Effet d'image"); final ToggleGroup
groupEffect = new ToggleGroup(); for (Entry<String, Effect> effect :
effects) {
    RadioMenuItem itemEffect = new RadioMenuItem(effect.getKey());
    itemEffect.setUserData(effect.getValue()); itemEffect.setToggleGroup(groupEffect);
    menuEffect.getItems().add(itemEffect);

} final MenuItem noEffects = new MenuItem("Aucun effet");
noEffects.setDisable(true); noEffects.setOnAction(new
EventHandler<ActionEvent>() {
    public void handle(ActionEvent t) { pic.setEffect(null);
        groupEffect.getSelectedToggle().setSelected(false);
        noEffects.setDisable(true);

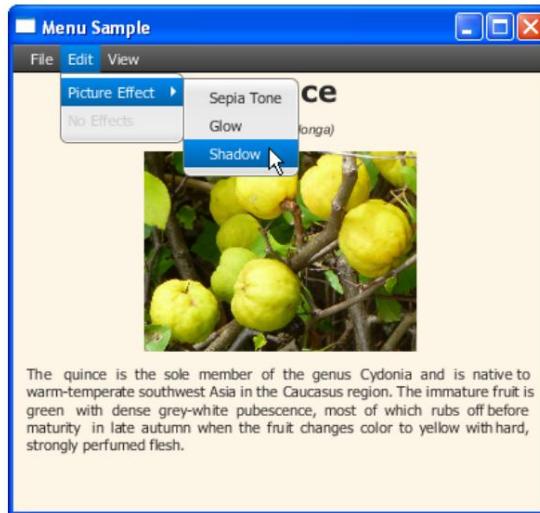
    }
});
```

groupEffect.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
 public void changé (ObservableValue<? étend Toggle> ov, Toggle old\_toggle,
 Toggle new\_toggle) { if (groupEffect.getSelectedToggle() != null) { Effect effect
 =
 (Effect) groupEffect.getSelectedToggle().getUserData();
 pic.setEffect(effect);
 **noEffects.setDisable(faux);** } else
 { noEffects.setDisable(true);

 }
});
†
menuEdit.getItems().addAll(menuEffect, noEffects);

Lorsqu'aucune des options RadioMenuItem n'est sélectionnée, l'élément de menu Aucun effet est désactivé, comme illustré à la [Figure 22-7](#). Lorsqu'un utilisateur sélectionne l'un des effets visuels, l'élément de menu Aucun effet est activé.

**Figure 22-7 L'élément du menu Effet est désactivé**



## Ajout de menus contextuels

Lorsque vous ne pouvez allouer aucun espace de votre interface utilisateur pour une fonctionnalité requise, vous pouvez utiliser un menu contextuel. Un menu contextuel est une fenêtre contextuelle qui apparaît en réponse à un clic de souris. Un menu contextuel peut contenir un ou plusieurs éléments de menu.

Dans l'application Menu Sample, définissez un menu contextuel pour l'image de la plante, afin que les utilisateurs puissent copier l'image.

Utilisez la classe `ContextMenu` pour définir le menu contextuel comme illustré dans l' [exemple 22-9](#).

### Exemple 22-9 Définition d'un menu contextuel

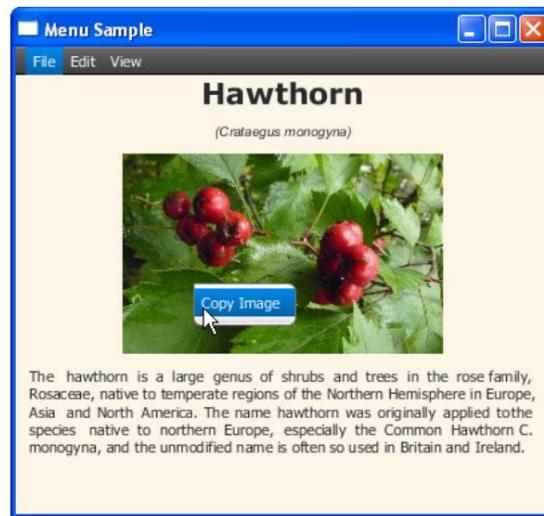
```
menu contextuel final cm = nouveau menu contextuel();
MenuItem cmItem1 = new MenuItem("Copier l'image");
cmItem1.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) { Clipboard
        clipboard = Clipboard.getSystemClipboard(); ClipboardContent content = new
        ClipboardContent(); content.putImage(pic.getImage()); presse-
        papiers.setContent(contenu);
    }
});

cm.getItems().add(cmItem1);
pic.addEventHandler(MouseEvent.MOUSE_CLIQUE,
    new EventHandler<MouseEvent>() {
        @Override public void handle(MouseEvent e) {
            if (e.getButton() == MouseButton.SECONDARY)
                cm.show(pic, e.getScreenX(), e.getScreenY());
        }
});
```

Lorsqu'un utilisateur clique avec le bouton droit sur l'objet ImageView, la méthode show est appelée pour que le menu contextuel active son affichage.

La méthode setOnAction définie pour l'élément Copier l'image du menu contextuel crée un objet Clipboard et ajoute l'image comme contenu. La Figure 22–8 capture un moment où un utilisateur sélectionne l'élément de menu contextuel Copier l'image.

**Figure 22–8 Utilisation du menu contextuel**



Vous pouvez essayer de copier l'image et de la coller dans un éditeur graphique.

Pour d'autres améliorations, vous pouvez ajouter d'autres éléments de menu au menu contextuel et spécifier différentes actions. Vous pouvez également créer un menu personnalisé à l'aide de la classe CustomMenuItem. Avec cette classe, vous pouvez intégrer un nœud arbitraire dans un menu et spécifier, par exemple, un bouton ou un curseur comme élément de menu.

#### Documentation relative à l'API

- ÿ Menus
- ÿ Élément de menu
- ÿ Élément de menu radio
- ÿ VérifierMenuItem
- ÿ Menu contextuel
- ÿ Élément de menu séparateur
- ÿ Élément de menu personnalisé

# 23

## 23Champ Mot de passe

Dans ce chapitre, vous découvrirez encore un autre type de contrôle de texte, le champ de mot de passe.

La classe `PasswordField` implémente un champ de texte spécialisé. Les caractères saisis par un utilisateur sont masqués par l'affichage d'une chaîne d'écho. La [Figure 23–1](#) montre un champ de mot de passe contenant un message d'invite.

Figure 23–1 Champ de mot de passe avec un message d'invite



### Création d'un champ de mot de passe

Une tâche d'entrée de gamme consiste à créer un champ de mot de passe à l'aide du code de l' [exemple 23–1](#).

#### Exemple 23–1 Création d'un champ de mot de passe

```
PasswordField passwordField = new PasswordField();  
passwordField.setPromptText("Votre mot de passe");
```

Pour votre interface utilisateur, vous pouvez accompagner le champ de mot de passe d'un message d'invite ou vous pouvez ajouter une étiquette de notification. Comme avec la classe `TextField`, la classe `PasswordField` fournit la méthode `setText` pour restituer une chaîne de texte dans le contrôle au démarrage de l'application. Cependant, la chaîne spécifiée dans `setText`

La méthode est masquée par les caractères d'écho dans le champ du mot de passe. Par défaut, le caractère d'écho est un astérisque. La [Figure 23–2](#) montre le champ du mot de passe avec le texte prédéfini.

Figure 23–2 Champ Mot de passe avec le texte défini



La valeur saisie dans le champ du mot de passe peut être obtenue via la méthode `getText`.

Vous pouvez traiter cette valeur dans votre application et définir la logique d'authentification appropriée.

**Évaluation du mot de passe**

Prenez un moment pour revoir dans l' [exemple 23–2](#) l'implémentation d'un champ de mot de passe que vous pouvez appliquer dans votre interface utilisateur.

**Exemple 23–2 Implémentation de la logique d'authentification**

message d'étiquette final = nouvelle étiquette("");

```
VBox vb = nouvelle VBox();
vb.setPadding(nouveaux inserts(10, 0, 0, 10));
vb.setSpacing(10); HBox hb = new HBox();
```

```
hb.setSpacing(10);
hb.setAlignment(Pos.CENTER_LEFT);
```

```
Label label = new Label("Mot de passe"); final
PasswordField pb = new PasswordField();
```

```
pb.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) { if (!
        pb.getText().equals("T2f$Ay!")) {
            message.setText("Votre mot de passe est incorrect !");
            message.setTextFill(Color.rgb(210, 39, 30)); } autre {
                message.setText("Votre mot de passe a été confirmé");
                message.setTextFill(Color.rgb(21, 117, 84));
            }
            pb.clear();
        }
});
```

```
hb.getChildren().addAll(label, pb);
vb.getChildren().addAll(hb, message);
```

La logique d'authentification du champ de mot de passe est définie à l'aide de la méthode `setOnAction`. Cette méthode est appelée lorsqu'un mot de passe est validé et crée un nouvel objet `EventHandler` pour traiter la valeur saisie. Si la valeur saisie est différente du mot de passe requis, le message correspondant apparaît en rouge comme illustré à la [Figure 23–3](#).

**Figure 23–3 Le mot de passe est incorrect**



Si la valeur saisie satisfait les critères prédéfinis, le message de confirmation s'affiche comme illustré à la [Figure 23–4](#).

**Figure 23–4 Le mot de passe est correct**



Pour des raisons de sécurité, il est recommandé d'effacer le champ du mot de passe après la saisie de la valeur. Dans l'[Exemple 23-2](#), une chaîne vide est définie pour le champ passwordField une fois l'authentification effectuée.

**Documentation relative à l'API**

ÿ Champ de mot de passe

ÿ TextInputControl

Évaluation du mot de passe

---

# 24

## 24 Sélecteur de couleurs

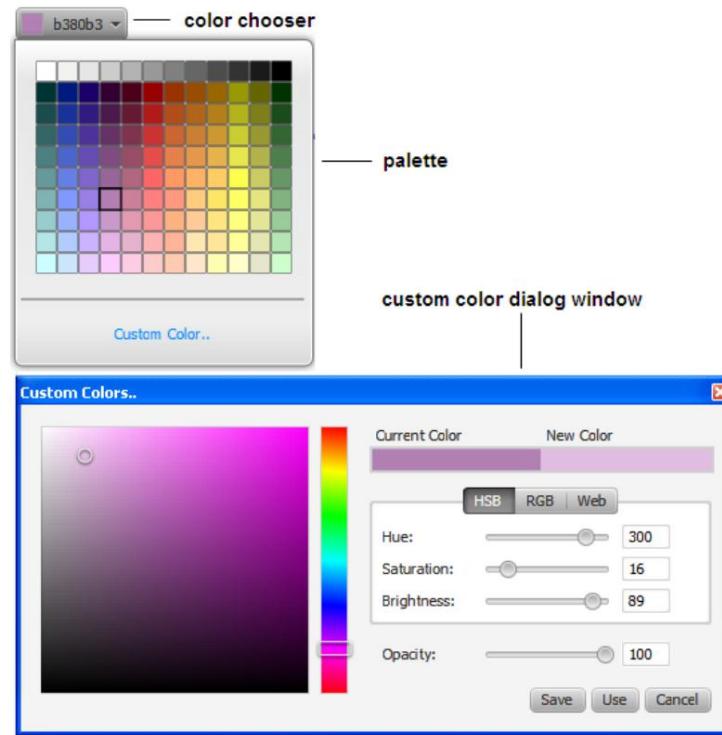
Ce chapitre décrit le contrôle ColorPicker, donne un aperçu de sa conception et explique comment l'utiliser dans vos applications JavaFX.

Le contrôle du sélecteur de couleurs dans le SDK JavaFX est un composant d'interface utilisateur typique qui permet aux utilisateurs de sélectionner une couleur particulière dans la gamme disponible ou de définir une couleur supplémentaire en spécifiant une combinaison RVB ou HSB.

### Aperçu de la conception

Le contrôle ColorPicker se compose du sélecteur de couleurs, de la palette de couleurs et de la fenêtre de dialogue de couleur personnalisée. [La Figure 24–1](#) montre ces éléments.

Figure 24–1 Éléments d'un contrôle de sélecteur de couleurs



## Sélecteur de couleurs

L'élément de sélection de couleurs du sélecteur de couleurs est la zone de liste déroulante avec l'indicateur de couleur activé et l'étiquette correspondante affichée en haut de la [Figure 24–1](#). L'indicateur de couleur présente la couleur actuellement sélectionnée.

L'implémentation du contrôle de sélecteur de couleurs permet trois aspects de l'élément de sélecteur de couleurs : bouton, bouton de menu partagé et zone de liste déroulante illustrés à la [Figure 24–2](#).

**Figure 24–2 Vues d'un sélecteur de couleurs**



L'apparence du bouton fournit un contrôle de bouton typique avec l'indicateur de couleur et l'étiquette. Dans l'apparence de bouton de menu divisé, la partie bouton du contrôle est séparée du menu déroulant. L'apparence de la liste déroulante est l'apparence par défaut du sélecteur de couleurs. Il a également un menu déroulant mais il n'est pas séparé de la partie bouton.

Pour appliquer l'un des looks, utilisez les classes CSS correspondantes. Pour plus d'informations sur la modification de l'apparence d'un sélecteur de couleurs, consultez [Modification de l'apparence d'un sélecteur de couleurs](#).

## Palette de couleurs

La palette de couleurs contient le jeu de couleurs prédéfini et le lien Couleur personnalisée qui pointe vers la boîte de dialogue Couleur personnalisée. L'aspect initial de la palette de couleurs est illustré à la [Figure 24–3](#).

**Figure 24–3 Aspect initial de la palette de couleurs**



Si une couleur personnalisée est déjà définie, cette couleur s'affiche dans la zone Couleur personnalisée de la palette de couleurs, comme illustré à [la Figure 24–4](#).

Figure 24–4 Palette de couleurs avec la zone de couleur personnalisée



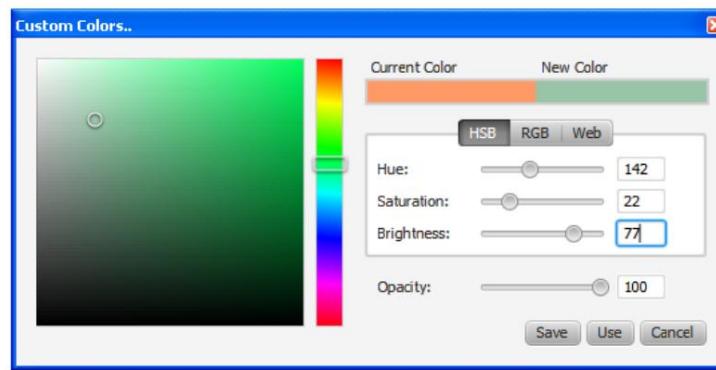
La palette de couleurs prend en charge la navigation à l'aide des touches Haut, Bas, Gauche et Droite.

L'ensemble de couleurs personnalisées ne peut pas être rechargé lorsque l'application redémarre, sauf s'il est enregistré dans l'application. Chaque couleur sélectionnée dans la palette ou dans la zone de couleur personnalisée est affichée dans l'indicateur de couleur du sélecteur de couleurs. L'étiquette du sélecteur de couleurs affiche la valeur de couleur Web hexadécimale correspondante.

#### Fenêtre de dialogue Couleur personnalisée

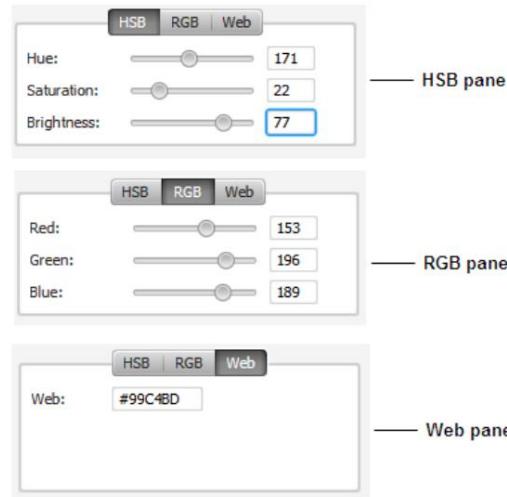
La boîte de dialogue Couleur personnalisée est une fenêtre modale qui peut être ouverte en cliquant sur le lien correspondant dans la palette de couleurs. Lorsque la fenêtre Couleur personnalisée s'ouvre, elle affiche les valeurs de la couleur actuellement affichée dans l'indicateur de couleur du sélecteur de couleurs. Les utilisateurs peuvent définir une nouvelle couleur en déplaçant le curseur de la souris sur la zone de couleur ou sur la barre de couleur verticale, illustrée à la Figure 24–5. Notez qu'à chaque fois qu'un utilisateur manipule avec le cercle dans la zone de couleur ou avec le rectangle dans la barre de couleur, la valeur de couleur est automatiquement affectée à la propriété correspondante du contrôle ColorPicker.

Figure 24–5 Fenêtre de dialogue Couleurs personnalisées



Une autre façon de définir une nouvelle couleur consiste à définir les valeurs HSB (Hue Saturation Brightness) ou RGB (Red Green Blue), ou à saisir explicitement la valeur de la couleur Web dans le champ correspondant. La Figure 24–6 montre trois volets pour les paramètres de couleur personnalisés.

Utiliser un sélecteur de couleurs

**Figure 24–6** Volets de réglage des couleurs dans la fenêtre de dialogue Couleurs personnalisées

Les utilisateurs peuvent également définir la transparence de la couleur personnalisée en déplaçant le curseur Opacité ou en tapant la valeur de 0 à 100.

Lorsque tous les paramètres sont définis et que la nouvelle couleur est spécifiée, les utilisateurs peuvent cliquer sur Utiliser pour l'appliquer ou sur Enregistrer pour enregistrer la couleur dans la zone de couleur personnalisée.

## Utiliser un sélecteur de couleurs

Utilisez la classe `ColorPicker` du SDK JavaFX pour créer un sélecteur de couleurs dans votre application JavaFX. Vous pouvez ajouter un sélecteur de couleurs directement à la scène de l'application, à un conteneur de mise en page ou à la barre d'outils de l'application. L'exemple 24–1 montre trois manières de déclarer un objet `ColorPicker`.

### Exemple 24–1 Création d'un contrôle de sélecteur de couleurs

```
// Constructeur vide, le champ apparaît avec la couleur par défaut, qui est le BLANC
ColorPicker colorPicker1 = nouveau ColorPicker();
// Constante de couleur définie comme la couleur actuellement sélectionnée
ColorPicker colorPicker2 = nouveau ColorPicker(Color.BLUE);
// Valeur de couleur Web définie comme la couleur actuellement sélectionnée
ColorPicker colorPicker3 = nouveau ColorPicker(Color.web("#ffccce6"));
```

Essayez l'exemple illustré dans l' [exemple 24–2](#) pour évaluer le contrôle `ColorPicker` en action.

### Exemple 24–2 Utilisation du contrôle `ColorPicker` pour modifier la couleur du composant de texte

```
importer javafx.application.Application;
import javafx.event.*;
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.geometry.Insets;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.*;
import javafx.stage.Stage;
```

```
la classe publique ColorPickerSample étend l'application {
    public static void main(String[] args) {
```

```

lancement(arguments);
}

@Passer outre
public void start(Stage stage) {
    stage.setTitle("ColorPicker");
    Scene scene = new Scene(new HBox(20), 400, 100);
    HBox box = (HBox) scene.getRoot();
    box.setPadding(nouveaux encarts(5, 5, 5, 5));

    ColorPicker final colorPicker = new ColorPicker();
    colorPicker.setValue(Color.CORAL);

    texte final text = new Text("Essayez le sélecteur de couleur!");
    text.setFont(Font.font("Verdane", 20));
    text.setFill(colorPicker.getValue());

    colorPicker.setOnAction(nouveau EventHandler() {
        public void handle(Event t) {
            text.setFill(colorPicker.getValue());
        }
    });
}

box.getChildren().addAll(colorPicker, text);

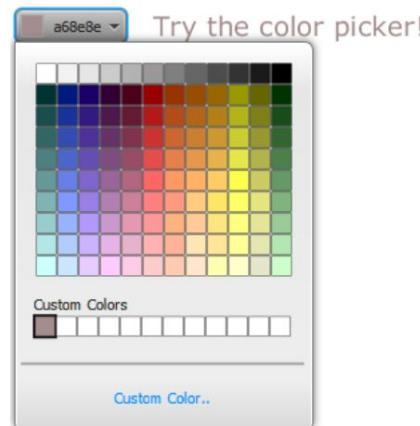
stage.setScene(scene);
spectacle();
}
}

```

Cet exemple crée un sélecteur de couleurs et définit son comportement lorsque la couleur est modifiée. La valeur Color obtenue via la méthode `getValue` de la classe `ColorPicker` est transmise à la méthode `setFill` de l'objet `Text`. C'est ainsi que la couleur sélectionnée est appliquée au "Essayez le sélecteur de couleurs!" texte.

Lorsque vous compilez et exécutez cet exemple, il produit la sortie illustrée à la [Figure 24–7](#). La figure capture le moment où une couleur personnalisée nouvellement créée est appliquée au composant Texte.

**Figure 24–7** ColorPicker et un composant de texte



De même, vous pouvez appliquer la couleur sélectionnée au Node graphique. [L'exemple 24–3](#) montre comment utiliser un sélecteur de couleurs pour modéliser un t-shirt.

### **Exemple 24–3 Utilisation de ColorPicker pour modifier la couleur d'un objet graphique**

```
importer javafx.application.Application; import
javafx.event.Event; import javafx.event.EventHandler;
importer javafx.scene.Scene; importer
javafx.scene.control.ColorPicker; importer
javafx.beans.value.ChangeListener; importez
javafx.beans.value.ObservableValue; importer
javafx.scene.control.ComboBox; importer
javafx.scene.control.ToolBar; importer
javafx.scene.effect.DropShadow; importer
javafx.scene.image.Image; importer
javafx.scene.image.ImageView; importer
javafx.scene.layout.StackPane; importer
javafx.scene.layout.VBox; importer javafx.scene.paint.Color;
importer javafx.scene.shape.SVGPath; importer
javafx.stage.Stage;
```

```
la classe publique ColorPickerSample étend l'application {

    Logo ImageView = nouvelle ImageView(
        nouvelle Image(getClass().getResourceAsStream("OracleLogo.png"))
    );

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

    }

    @Passer autre
    public void start(Stage stage) {
        stage.setTitle("ColorPicker");

        Scene scene = new Scene(new VBox(20), 300, 300);
        scene.setFill(Color.web("#ccffcc")); Boîte VBox =
(VBox) scène.getRoot();

        ToolBar tb = new ToolBar();
        box.getChildren().add(tb);

        final ComboBox logoSamples = new ComboBox();
        logoSamples.getItems().addAll( "Oracle",
            "Java",
            "JavaFX",
            "Coupe");
        logoSamples.setValue("Oracle");

        logoSamples.valueProperty().addListener(new ChangeListener<String>() {
            @Passer autre
            public void changé (ObservableValue ov, String t, String t1) {
                logo.setImage( nouvelle
                    Image(getClass().getResourceAsStream(t1+"Logo.png"))
                );
            }
        });
    }
}
```

```

ColorPicker final colorPicker = new ColorPicker();
tb.getItems().addAll(logoSamples, colorPicker);

Pile StackPane = new StackPane();
box.getChildren().add(pile);

chemin SVG final svg = nouveau chemin SVG();
svg.setContent("M70,50 L90,50 L120,90 L150,50 L170,50"
    + "L210,90 L180,120 L170,110 L170,200 L70,200 L70,110 L60,120 L30,90"
    + "L70,50");
svg.setStroke(Color.DARKGREY);
svg.setStrokeWidth(2);
svg.setEffect(new DropShadow());
svg.setFill(colorPicker.getValue());
stack.getChildren().addAll(svg, logo);

colorPicker.setOnAction(nouveau EventHandler() {
    public void handle(Event t) {
        svg.setFill(colorPicker.getValue());
    }
});

stage.setScene(scene);
spectacle();
}
}

```

Dans cet exemple, la couleur sélectionnée dans le sélecteur de couleurs et obtenue via la méthode `getValue` est appliquée à l'objet `SVGPath`. Cet exemple produit la sortie illustrée à la [Figure 24–8](#)

**Figure 24–8 Exemple d'application ColorPicker**



Lorsque vous travaillez avec un sélecteur de couleurs, vous pouvez obtenir les couleurs personnalisées créées en appelant la méthode `getCustomColors()`. Elle renvoie une `ObservableList` des objets `Color` correspondant aux couleurs créées. Vous ne pouvez pas les télécharger dans un sélecteur de couleurs au démarrage de l'application. Cependant, vous pouvez définir l'une des couleurs personnalisées comme valeur `ColorPicker` (illustrée dans [l'Exemple 24–4](#)).

**Exemple 24–4 Obtention des couleurs personnalisées**

```
ObservableList<Color> customColors = colorPicker.getCustomColors();
colorPicker.setValue(customColors.get(index));
```

**Modification de l'apparence d'un sélecteur de couleurs**

L'apparence par défaut du contrôle du sélecteur de couleurs est définie par la classe com.sun.javafx.scene.control.skin.ColorPickerSkin. Pour appliquer un skin alternatif aux sélecteurs de couleurs dans votre application JavaFX, redéfinissez le -fx-skin propriété de la classe CSS color-picker comme illustré dans l' [Exemple 24–5](#).

**Exemple 24–5 Définition d'un habillage alternatif pour un sélecteur de couleurs**

```
.pipette à couleurs {
    -fx-skin: "PersonnaliséSkin";
}
```

Utilisez les classes CSS split-button et arrow-button pour modifier l'apparence d'un contrôle de sélecteur de couleur dans le code JavaFX, comme illustré dans l' [exemple 24–6](#).

**Exemple 24–6 Définition de l'apparence d'un sélecteur de couleurs**

```
//Définit le bouton de menu partagé
colorPicker.getStyleClass().add("split-button");
// Définit le bouton
colorPicker.getStyleClass().add("bouton");
```

Vous pouvez également modifier le style par défaut d'un sélecteur de couleurs et personnaliser ses éléments avec diverses classes CSS définies dans la feuille de style caspienne. Pour afficher ce fichier, accédez au répertoire \rt\lib sous le répertoire dans lequel le SDK JavaFX est installé. Utilisez la commande suivante pour extraire la feuille de style du fichier JAR : jar -xf jfxrt.jar com/sun/javafx/scene/control/skin/caspian/caspian.css.

Voir Habiller les applications JavaFX avec CSS pour plus d'informations sur les classes et les propriétés CSS. L'[exemple 24–7](#) montre comment modifier l'arrière-plan par défaut et l'étiquette d'un sélecteur de couleurs.

**Exemple 24–7 Modification de l'apparence par défaut d'un sélecteur de couleurs**

```
.pipette à couleurs {
    -fx-background-color: #669999; -fx-rayon
    de fond: 0 15 15 0;
}
.color-picker .color-picker-label .text {
    -fx-fill: #ccffcc;
```

Ajoutez ces styles au fichier ControlStyle.css et activez les feuilles de style dans l'application JavaFX en utilisant la ligne de code suivante : scene.getStylesheets().add("colorpickersample/ControlStyle.css") ;, puis compilez et exécutez le ColorPickerSample. Le sélecteur de couleurs devrait changer d'apparence, comme illustré à la [Figure 24–9](#).

**Figure 24–9 Apparence modifiée d'un sélecteur de couleurs**

Notez que la classe `ColorPicker` est une extension de la classe `ComboBoxBase` et hérite de ses propriétés CSS. Vous pouvez définir de nouveaux styles pour le style CSS `combo-box-base` afin d'unifier la zone de liste déroulante et le sélecteur de couleurs dans l'application `ColorPickerSample`. Remplacez les styles du fichier `ControlStyle.css` par les styles illustrés dans l'[Exemple 24–8](#).

**Exemple 24–8 Définition des styles de base Combo-Box**

```
.tool-bar:horizontale {  
    -fx-background-color: #b3e6b3;  
}  
  
.combo-box-base {  
    -fx-background-color: null;  
}  
  
.combo-box-base:hover {  
    -fx-effect: dropshadow( box, rgba(0,0,0,0.6) , 8, 0.0  
        , 0 , 0 );  
}
```

Lorsque vous compilez et exécutez l'application `ColorPickerSample` avec les styles appliqués, la zone de liste déroulante et le sélecteur de couleurs ressemblent à ceux de la [Figure 24–10](#).

Modification de l'apparence d'un sélecteur de couleurs

Figure 24–10 Style unifié de la zone de liste déroulante et du sélecteur de couleurs



### Documentation relative à l'API

- ÿ Sélecteur de couleurs
- ÿ ComboBoxBase

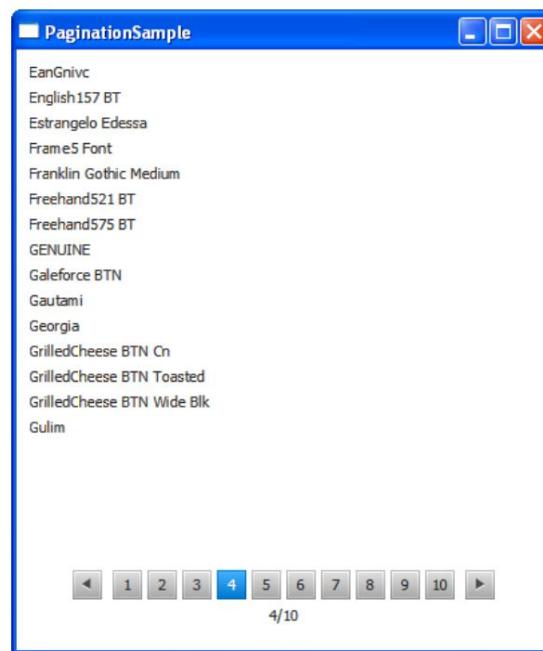
# 25

## Contrôle de la pagination

Ce chapitre explique comment ajouter un contrôle de pagination à votre application JavaFX. Il enseigne comment ajouter un contrôle de pagination à votre application, gérer ses éléments de page et styliser les éléments du contrôle avec des styles CSS.

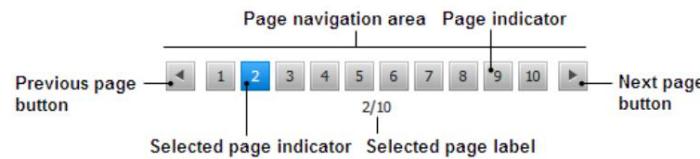
Le contrôle de pagination utilisé pour parcourir plusieurs pages de contenu divisées en parties plus petites. Les utilisations typiques incluent la navigation dans les messages électroniques dans une boîte aux lettres ou le choix parmi les résultats de recherche. Dans les appareils tactiles, un contrôle de pagination peut être utilisé pour parcourir les pages individuelles d'un article ou pour naviguer entre les écrans. [La Figure 25–1](#) montre un contrôle de pagination qui affiche les polices disponibles dans un système d'exploitation.

**Figure 25–1 Contrôle de la pagination**



### Création d'un contrôle de pagination

Un contrôle de pagination se compose du contenu de la page et des zones de navigation de la page. La zone de contenu de la page restitue et présente le contenu conformément à la logique de l'application. La zone de navigation Page contient un contrôle préfabriqué pour prévisualiser une partie particulière du contenu. [La Figure 25–2](#) montre la structure et les éléments de base de la navigation Région.

**Figure 25–2 Zone de navigation d'un contrôle de pagination**

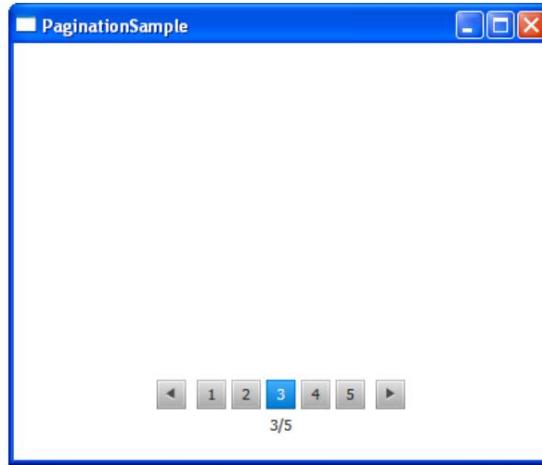
Vous pouvez parcourir les pages en cliquant sur un indicateur de page particulier ou en cliquant sur les boutons Page suivante et Page précédente. Lorsque la première page est sélectionnée, le bouton Page précédente est désactivé. De même, lorsque le dernier indicateur de navigation est sélectionné, le bouton Page suivante est désactivé.

L'API JavaFX SDK fournit la classe `Pagination` pour ajouter le contrôle de pagination à votre application JavaFX. [L'exemple 25–1](#) montre trois constructeurs disponibles de la classe `Pagination`.

### **Exemple 25–1 Trois constructeurs de la classe Pagination**

```
//Crée un champ Pagination avec un nombre de pages INDÉTERMINÉ //et l'index de la page courante
égal à zéro
pagination1 = nouvelle Pagination();
//Crée un champ Pagination de 5 pages
//et l'index de la page courante égal à zéro
pagination2 = nouvelle Pagination(5);
//Crée un champ Pagination de 5 pages
//et l'index actuellement sélectionné égal à 2
pagination3 = nouvelle Pagination(5, 2);
```

La commande `pagination3` de l' [Exemple 25–1](#) est illustrée à la [Figure 25–3](#).

**Figure 25–3 Contrôle de la pagination sans le contenu**

Notez que les index de page commencent par 0. Par conséquent, pour commencer avec la troisième page sélectionnée, vous devez définir `currentPageIndexProperty` sur 2.

Les pages du champ `pagination3` sont vides, car aucun contenu n'est ajouté au champ.

Vous ne pouvez pas ajouter d'éléments directement au contrôle de pagination, cela nécessite de définir une fabrique de pages. Utilisez la méthode setPageFactory de la classe Pagination pour définir le contenu de la page en implémentant une fabrique de page.

## Implémentation des fabriques de pages

Le setPageFactory est utilisé pour définir une fabrique de pages pour le contrôle de pagination. Le développeur de l'application crée une méthode de rappel et définit la fabrique de page de pagination pour utiliser ce rappel. La méthode de rappel est appelée lorsqu'une page a été sélectionnée. Il charge et renvoie le contenu de la page sélectionnée. La valeur nulle doit être renvoyée si l'index de page sélectionné n'existe pas. [L'exemple 25–2](#) crée un contrôle de pagination avec 28 pages et remplit les pages avec les résultats de la recherche, en allouant huit éléments par page.

### Exemple 25–2 Ajout d'hyperliens à un contrôle de pagination

```
importer javafx.application.Application; importer
javafx.scene.Scene; importer
javafx.scene.control.Pagination; importer
javafx.scene.Node; importer javafx.scene.control.Hyperlink;
importer javafx.scene.control.Label; importer
javafx.scene.layout.AnchorPane; importer
javafx.scene.layout.VBox; importer javafx.stage.Stage;
importer javafx.util.Callback;

public class PaginationSample étend Application { private Pagination
pagination;

public static void main(String[] args) lance une exception { launch(args);

}

public int élémentsParPage() { return
8;
}

public VBox createPage(int pageIndex) {
Boîte VBox = nouvelle VBox(5);
int page = pageIndex * élémentsParPage(); for (int
je = page; je < page + élémentsParPage(); i++) {
élément VBox = nouvelle VBox();
lien hypertexte = nouveau lien hypertexte("Item " + (i+1));
link.setVisited(true);
Texte de l'étiquette = new Label("Résultats de la recherche\npour " + lien.getText());
element.getChildren().addAll(lien, texte); box.getChildren().add(element);

}
boîte de retour;
}

@Passer autre
public void start (étape finale) lance une exception {
pagination = nouvelle Pagination(28, 0);
pagination.setStyle("-fx-border-color:rouge;");
pagination.setPageFactory(new Callback<Integer, Node> {
@Passer autre
```

```

        appel de nœud public (Integer pageIndex) {
            return createPage(pageIndex);
        }
    });

AnchorPane anchor = new AnchorPane();
AnchorPane.setTopAnchor(pagination, 10.0);
AnchorPane.setRightAnchor(pagination, 10.0);
AnchorPane.setBottomAnchor(pagination, 10.0);
AnchorPane.setLeftAnchor(pagination, 10.0);
ancre.getChildren().addAll(pagination);
Scene scene = new Scene(ancre);
stage.setScene(scene);
stage.setTitle("PaginationSample");
spectacle();
}
}

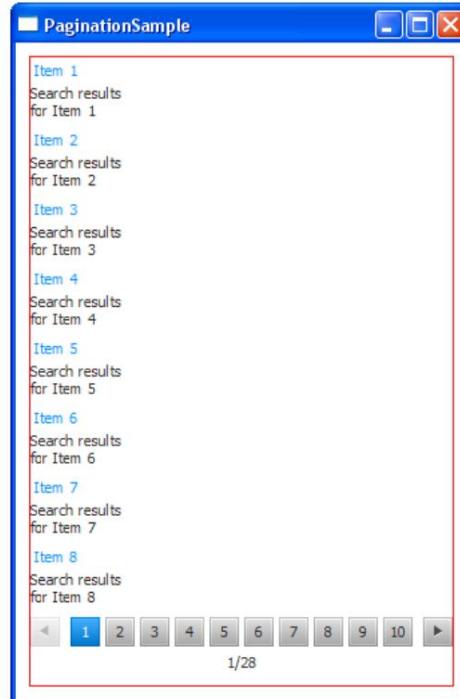
```

Le nombre de pages et la page sélectionnée sont définis dans le constructeur de la classe `Pagination`. Vous pouvez également créer un contrôle `Pagination` et définir ensuite le nombre de pages et l'index de la page sélectionnée à l'aide des méthodes `setPageCount` et `setCurrentPageIndex`.

Le contenu du contrôle `Pagination` est déclaré dans la méthode `createPage` qui sert de fabrique de pages et est appelé par la méthode `setPageFactory`. La page de création crée les paires d'hyperliens et les étiquettes correspondantes, et les organise verticalement, en définissant un intervalle de cinq pixels entre les éléments.

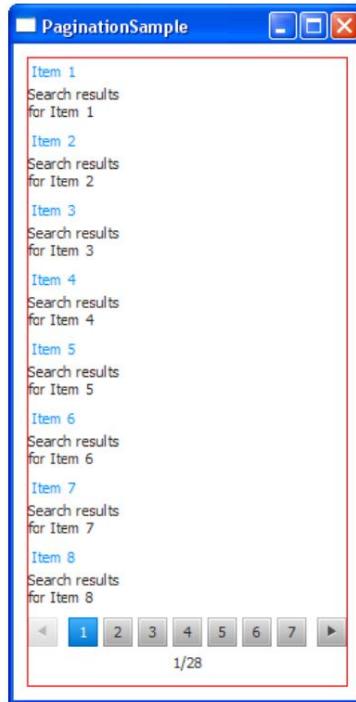
Lorsque vous compilez et exécutez [l'Exemple 25–2](#), vous devriez voir la sortie illustrée à la [Figure 25–4](#).

**Figure 25–4 Utilisation d'un contrôle de pagination pour prévisualiser les résultats de la recherche**



L'implémentation actuelle du contrôle Pagination affiche 10 indicateurs de page si le nombre de pages dépasse 10. Pour définir une autre valeur pour les indicateurs de page affichés, utilisez la méthode setMaxPageIndicatorCount de la classe Pagination . Par exemple, ajoutez la ligne suivante à l' [Exemple 25–2](#) pour afficher sept indicateurs de page : pagination.setMaxPageIndicatorCount(7);. La Figure 25–5 illustre l'application PaginationSample une fois la modification appliquée.

**Figure 25–5 Modification du nombre d'indicateurs de page**



L'[Exemple 25–3](#) montre une autre utilisation du contrôle de pagination. L'application restitue les fragments de texte, un par page. Le nombre de fragments est de cinq et le nombre de pages déclarées du contrôle de pagination est de 28. Pour éviter une condition `ArrayIndexOutOfBoundsException`, ajoutez la vérification d'index de page (surlignée en gras dans l' [Exemple 25–3](#)) et faites en sorte que la méthode de rappel renvoie null lorsque le nombre de pages dépasse cinq.

#### Exemple 25–3 Ajout d'extraits de texte à un contrôle de pagination

```
importer javafx.application.Application;
importer javafx.scene.Scene;
importer javafx.scene.control.Pagination;
importer javafx.scene.Node;
importer javafx.scene.control.TextArea;
importer javafx.scene.layout.AnchorPane;
importer javafx.scene.layout.VBox;
importer javafx.stage.Stage;
importer javafx.util.Callback;

la classe publique PaginationSample étend l'application {

    Pagination privée pagination;
    chaîne finale [] textPages = nouvelle chaîne [] {
        « La pomme est le fruit à pépins du pommier, espèce Malus »
        + "domestica dans la famille des roses (Rosaceae). C'est l'une des plus
        " "
```

+ "les fruits des arbres largement cultivés, et le plus connu des " + "les nombreux membres du genre Malus qui sont utilisés par les humains. "  
+ "L'arbre est originaire d'Asie occidentale, où son ancêtre sauvage, + "l'Alma, se trouve encore aujourd'hui.", "L'aubépine est un grand genre d'arbustes et d'arbres de la famille des roses",  
+ "Rosaceae, originaire des régions tempérées de l'hémisphère Nord " + " en Europe, en Asie et en Amérique du Nord. Le nom d'aubépine était + "appliqué à l'origine à l'espèce originaire du nord de l'Europe " + ", en particulier l'Aubépine commune C. monogyna, et le nom " + " non modifié est souvent utilisé en Grande-Bretagne et en Irlande.",  
"Le lierre est une plante à fleurs de la famille des raisins (Vitacées) originaire d'Asie orientale au Japon, en + " Corée et dans le nord et l'est de la Chine."  
+ "C'est une vigne ligneuse à feuilles caduques atteignant 30 m de hauteur ou plus" + "et dotée d'un support approprié, se fixant au moyen de nombreuses petites " + "villes ramifiées terminées par des disques collants.", "Le coing est le seul membre de la genre Cydonia et est originaire de + "l'Asie du sud-ouest tempérée chaude dans la région du Caucase. Le " + "fruit immature est vert avec une pubescence gris-blanc dense, dont la plupart + "déteint avant maturité à la fin de l'automne lorsque le fruit " + "change de couleur en jaune avec une chair dure et fortement parfumée.", "Aster (syn. Diplopappus Cass.) est un genre de plantes à fleurs " + " de la famille des Astéracées. Le genre contenait autrefois près de 600 " + " espèces en Eurasie et en Amérique du Nord, mais après des recherches morphologiques " + " et moléculaires sur le genre au cours des années 1990 , il a été + "décidé que les espèces nord-américaines seraient mieux traitées dans une + "série d'autres genres apparentés. Après cette scission, il y a + "environ 180 espèces au sein du genre, toutes sauf une étant confinées " + "à l'Eurasie".  
"

});

```
public static void main(String[] args) lance une exception { launch(args);
```

```
}
```

```
public int itemsPerPage() { return 1;
```

```
}
```

```
public VBox createPage(int pageIndex) { VBox box = new
    VBox(5); int page = pageIndex * élémentsParPage();
    for (int i = page; i < page + itemsPerPage(); i++) { TextArea
        text = new TextArea(textPages[i]); text.setWrapText(true);
        box.getChildren().add(texte);
```

```
}
```

```
boîte de retour;
```

```
}
```

```
@Passer outre
```

```
public void start (étape finale) lance une exception {
    pagination = nouvelle Pagination(28, 0);
    pagination.setStyle("-fx-border-color:rouge;");
    pagination.setPageFactory(new Callback<Integer, Node>() {
```

```
@Passer outre
```

```
appel de nœud public (Integer pageIndex) {
```

```
if (pageIndex >= textPages.length) {
```

```
renvoie nul;
```

```
autre {
```

```

        return createPage(pageIndex);
    }
}

};

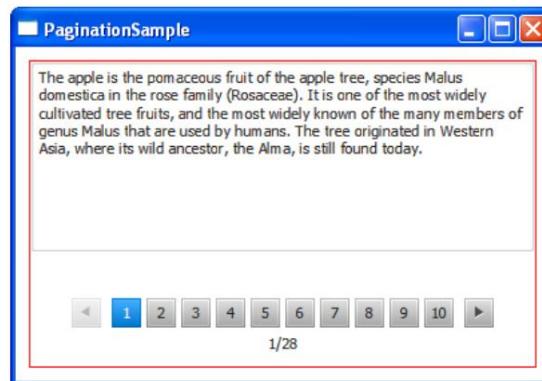
AnchorPane anchor = new AnchorPane();
AnchorPane.setTopAnchor(pagination, 10.0);
AnchorPane.setRightAnchor(pagination, 10.0);
AnchorPane.setBottomAnchor(pagination, 10.0);
AnchorPane.setLeftAnchor(pagination, 10.0);
ancre.getChildren().addAll(pagination); Scene scene = new
Scene(ancre, 400, 250); stage.setScene(scene);
stage.setTitle("PaginationSample"); spectacle();
}

}
}

```

Lorsque vous compilez et exécutez l' [Exemple 25–3](#), vous verrez la sortie illustrée à la [Figure 25–6](#).

**Figure 25–6 Rendu des fragments de texte dans un contrôle de pagination**



Dans certains cas, vous ne pouvez pas définir le nombre exact d'éléments à afficher et, par conséquent, le nombre de pages dans un contrôle de pagination. Dans de telles situations, vous pouvez inclure une ligne de code qui calcule le nombre de pages dans le constructeur de l' objet Pagination . [L'exemple 25–4 génère](#) une liste de polices système et calcule le nombre de pages comme la longueur du tableau de polices divisée par le nombre d'éléments par page.

#### **Exemple 25–4 Ajout de contenu d'une taille indéterminée**

```

importer javafx.application.Application; importer
javafx.scene.Scene; importer
javafx.scene.control.Pagination; importer
javafx.scene.Node; importer javafx.scene.control.Label;
importer javafx.scene.layout.AnchorPane; importer
javafx.scene.layout.VBox; importer javafx.scene.text.Font;
importer javafx.stage.Stage; importer javafx.util.Callback;

```

la classe publique PaginationSample étend l'application {

Pagination privée pagination;

```

polices String[] = new String[]{}};

public static void main(String[] args) lance une exception { launch(args);

}

public int élémentsParPage() { return 15;

}

public VBox createPage(int pageIndex) {
    Boîte VBox = nouvelle VBox(5);
    int page = pageIndex * élémentsParPage(); for (int je =
page; je < page + élémentsParPage(); i++) {
        Police de l'étiquette = nouvelle étiquette (fonts
[i]); box.getChildren().add(police);
    }
    boîte de retour;
}

@Passer outre
public void start (étape finale) lance une exception {
    fonts = Font.getFamilies().toArray(fonts);

pagination = new Pagination(fonts.length/itemsPerPage(), 0); pagination.setStyle("-fx-border-color:rouge;"); pagination.setPageFactory(new Callback<Integer, Node>() {

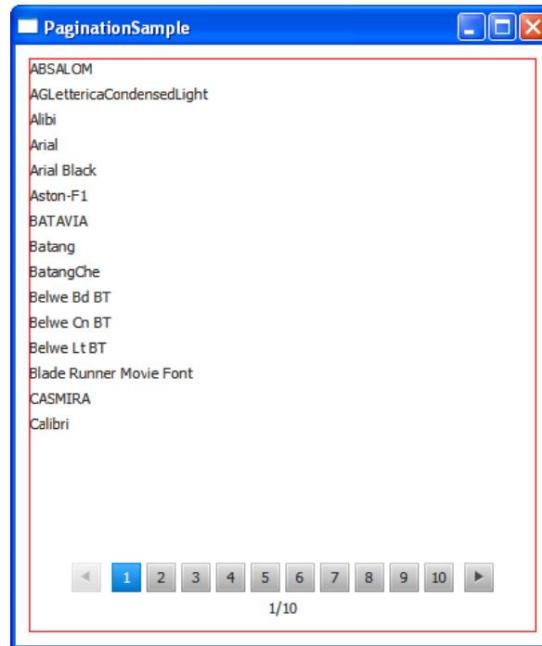
    @Passer outre
    public Node call(Integer pageIndex) { return
        createPage(pageIndex);
    }
});

AnchorPane anchor = new AnchorPane();
AnchorPane.setTopAnchor(pagination, 10.0);
AnchorPane.setRightAnchor(pagination, 10.0);
AnchorPane.setBottomAnchor(pagination, 10.0);
AnchorPane.setLeftAnchor(pagination, 10.0);
ancr.getChildren().addAll(pagination); Scene scene = new
Scene(ancr, 400, 450);
stage.setScene(scene);
stage.setTitle("PaginationSample"); spectacle();

}
}

```

Lorsque vous compilez et exécutez cet exemple, il produit la fenêtre d'application illustrée à la Figure 25–7.

**Figure 25–7 Utilisation d'un contrôle de pagination pour rendre les polices système**

## Styliser un contrôle de pagination

Vous pouvez personnaliser le contrôle de pagination pour afficher des indicateurs de page à puces au lieu d'indicateurs de page numériques en définissant la classe de style `STYLE_CLASS_BULLET`. De plus, vous pouvez modifier les styles de pagination par défaut dans la feuille de style caspienne .

L'[exemple 25–5](#) montre quelques styles alternatifs pour les éléments visuels du contrôle de pagination de l'[exemple 25–4](#).

### Exemple 25–5 Styles modifiés du contrôle de pagination

```
.pagination {
    -fx-border-color: #0E5D79;
}

.pagination .page {
    -fx-background-color: #DDF1F8;
}

.pagination .pagination-control {
    -fx-background-color: #C8C6C6;
}

.pagination .pagination-control .bullet-bouton {
    -fx-background-color: transparent, #DDF1F8, #0E5D79, blanc, blanc;
}

.pagination .pagination-control .bullet-bouton:selectionné {
    -fx-background-color: transparent, #DDF1F8, #0E5D79, blanc, #0E5D79;
}

.pagination .pagination-control .left-arrow, .right-arrow{
    -fx-background-color: #DDF1F8, #0E5D79;
}
```

L'exemple 25–6 applique ces styles au contrôle de pagination et définit le style de puce pour les indicateurs de page.

### Exemple 25–6 Activation du style de contrôle de pagination modifié dans PaginationSample Application

```
importer javafx.application.Application; importer
javafx.scene.Scene; importer
javafx.scene.control.Pagination; importer
javafx.scene.Node; importer javafx.scene.control.Label;
importer javafx.scene.layout.AnchorPane; importer
javafx.scene.layout.VBox; importer javafx.scene.text.Font;
importer javafx.stage.Stage; importer javafx.util.Callback;

la classe publique PaginationSample étend l'application {

    Pagination privée pagination;
    polices String[] = new String[]{};

    public static void main(String[] args) lance une exception { launch(args);

    }

    public int élémentsParPage() { return 15;

    }

    public VBox createPage(int pageIndex) {
        Boîte VBox = nouvelle VBox(5);
        int page = pageIndex * élémentsParPage(); for (int je =
        page; je < page + élémentsParPage(); i++) {
            Police de l'étiquette = nouvelle étiquette (fonts
            [i]); box.getChildren().add(police);
        }
        boîte de retour;
    }

    @Passer autre
    public void start (étape finale) lance une exception {
        fonts = Font.getFamilies().toArray(fonts);

        pagination = new Pagination(fonts.length/itemsPerPage(), 0);
pagination.getStyleClass().add(Pagination.STYLE_CLASS_BULLET);
        pagination.setPageFactory(new Callback<Integer, Node>() {

            @Passer autre
            public Node call(Integer pageIndex) { return
                createPage(pageIndex);
            }
        });
    }

    AnchorPane anchor = new AnchorPane();
    AnchorPane.setTopAnchor(pagination, 10.0);
    AnchorPane.setRightAnchor(pagination, 10.0);
    AnchorPane.setBottomAnchor(pagination, 10.0);
    AnchorPane.setLeftAnchor(pagination, 10.0);
}
```

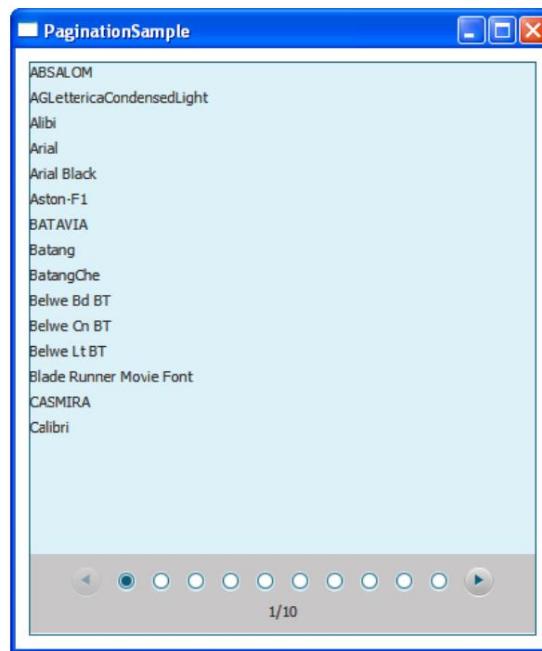
```

        ancre.getChildren().addAll(pagination);
        Scene scene = new Scene(ancre, 400, 450);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        scene.getStylesheets().add("paginationsample/ControlStyle.css");
        spectacle();
    }
}

```

Lorsque vous appliquez les styles nouvellement définis à l'application PaginationSample, que vous la compilez et l'exécutez, la fenêtre de l'application illustrée à la [Figure 25–8](#) s'affiche.

**Figure 25–8 PaginationSample avec des indicateurs de page à puces et les nouveaux styles CSS Appliqués**



En plus des styles appliqués, vous pouvez envisager les styles suivants pour modifier l'apparence du contrôle de pagination dans vos applications :

- ÿ -fx-max-page-indicator-count — Définit le nombre maximal d'indicateurs de page.
- ÿ -fx-arrows-visible — Bascule la visibilité des flèches des boutons Suivant et Précédent, vrai par défaut.
- ÿ -fx-tooltip-visible — Bascule la visibilité de l'info-bulle de l'indicateur de page, vrai par défaut.
- ÿ -fx-page-information-visible — Active/désactive la visibilité des informations de la page, true par défaut.
- ÿ -fx-page-information-alignment — Définit l'alignement des informations de la page.

#### Documentation relative à l'API

ÿ Mise en page

ÿ VBox

ÿ AnchorPane

# 26

---

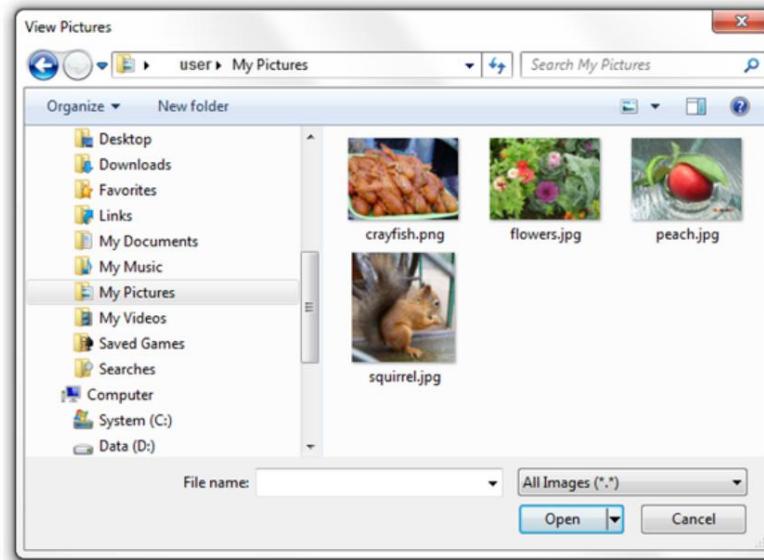
## Sélecteur de fichiers

Ce chapitre explique comment utiliser la classe `FileChooser` pour permettre aux utilisateurs de naviguer dans le système de fichiers. Les exemples fournis dans ce chapitre expliquent comment ouvrir un ou plusieurs fichiers, configurer une fenêtre de dialogue de sélection de fichiers et enregistrer le contenu de l'application.

Contrairement aux autres classes de composants d'interface utilisateur, la classe `FileChooser` n'appartient pas au package `javafx.scene.controls`. Cependant, cette classe mérite d'être mentionnée dans le tutoriel JavaFX UI Controls, car elle prend en charge l'une des fonctions typiques de l'application GUI : la navigation dans le système de fichiers.

La classe `FileChooser` se trouve dans le package `javafx.stage` avec les autres éléments graphiques racine de base, tels que `Stage`, `Window` et `Popup`. La fenêtre Afficher les images de la Figure 26–1 est un exemple de la boîte de dialogue de sélection de fichiers de Windows.

Figure 26–1 Exemple de fenêtre de sélection de fichiers



### Ouverture de fichiers

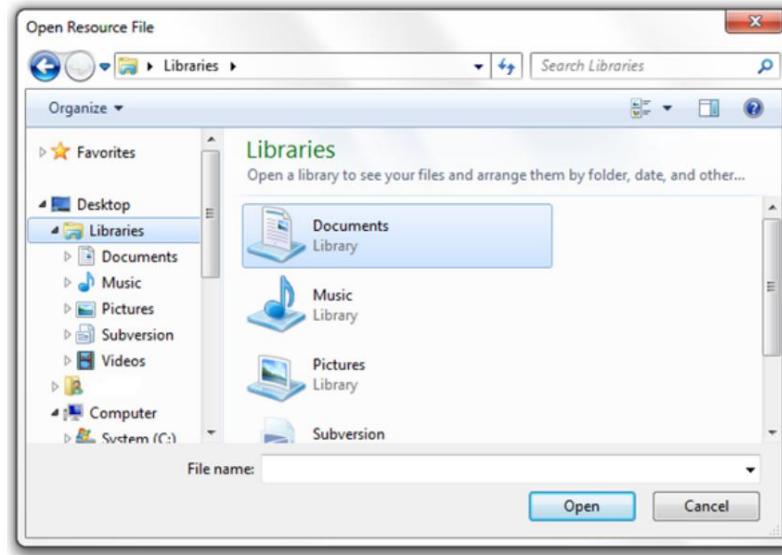
Un sélecteur de fichier peut être utilisé pour invoquer une fenêtre de dialogue ouverte pour sélectionner un seul fichier ou plusieurs fichiers, et pour activer une fenêtre de dialogue d'enregistrement de fichier. Pour afficher un sélecteur de fichier, vous utilisez généralement la classe `FileChooser`. L' [exemple 26–1](#) fournit la manière la plus simple d'activer un sélecteur de fichier dans votre application.

**Exemple 26–1 Affichage d'un sélecteur de fichiers**

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Ouvrir le fichier de ressources");
fileChooser.showOpenDialog(étape);
```

Une fois le code de l' [Exemple 26–1](#) ajouté à une application JavaFX, la boîte de dialogue de sélection de fichier apparaît immédiatement au démarrage de l'application, comme illustré à la [Figure 26–2](#).

**Figure 26–2 Sélecteur de fichier simple**



---

**Remarque :** la [Figure 26–2](#) montre le sélecteur de fichiers dans Windows. Lorsque vous ouvrez des sélecteurs de fichiers dans d'autres systèmes d'exploitation qui prennent en charge cette fonctionnalité, vous recevez des fenêtres alternatives. La [Figure 26–3](#) et la [Figure 26–4](#) montrent des exemples de fenêtres de sélection de fichiers sous Linux et Mac OS.

---

Figure 26–3 Fenêtre du sélecteur de fichiers sous Linux

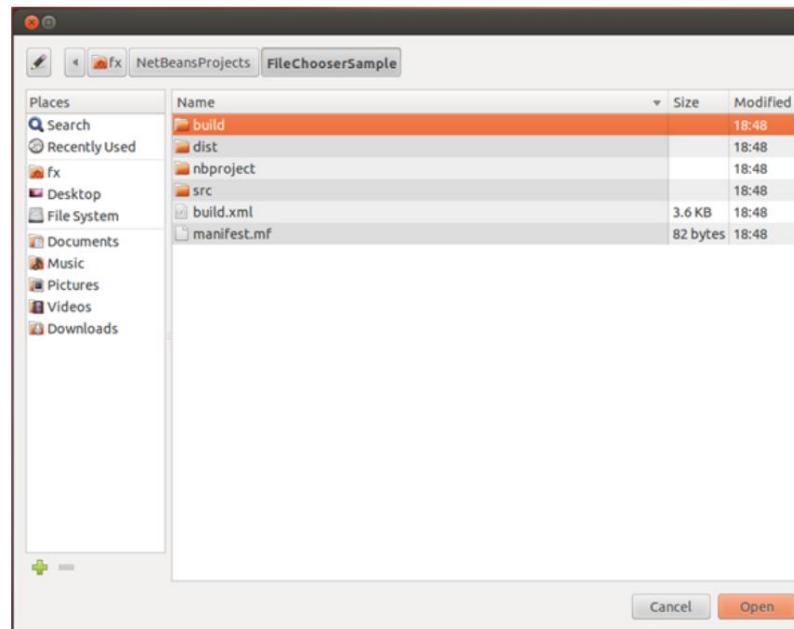
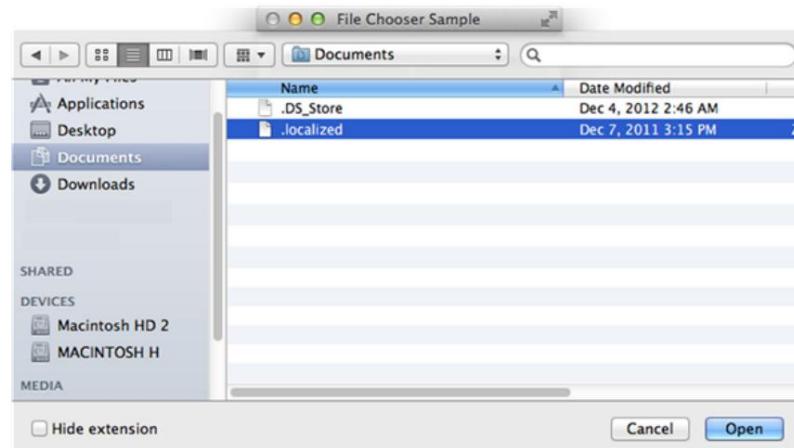


Figure 26–4 Fenêtre de sélection de fichiers sous Mac OS



Bien que dans l'exemple précédent, le sélecteur de fichier s'affiche automatiquement au démarrage de l'application, une approche plus typique consiste à invoquer un sélecteur de fichier en sélectionnant l'élément de menu correspondant ou en cliquant sur un bouton dédié. Dans ce didacticiel, vous créez une application qui permet à un utilisateur de cliquer sur un bouton et d'ouvrir une ou plusieurs images situées dans le système de fichiers. L'[exemple 26–2](#) montre le code de l'application FileChooserSample qui implémente cette tâche.

#### Exemple 26–2 Ouverture du sélecteur de fichier pour une sélection unique et multiple

```
importer java.awt.Desktop;
importer java.io.File;
import java.io.IOException;
importer java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```

importer javafx.application.Application; import
javafx.event.ActionEvent; import javafx.event.EventHandler;
importer javafx.geometry.Insets; importer javafx.scene.Scene;
importer javafx.scene.control.Button; import
javafx.scene.layout.GridPane; import
javafx.scene.layout.Pane; import javafx.scene.layout.VBox;
importer javafx.stage.FileChooser; importer javafx.stage.Stage;

```

La classe finale publique FileChooserSample étend l'application {

```

Bureau privé desktop = Desktop.getDesktop();

@Passer autre
début du vide public (étape finale) {
    stage.setTitle("Échantillon de sélecteur de fichiers");

    final FileChooser fileChooser = new FileChooser();

    bouton final openButton = new Button("Ouvrir une image..."); bouton final openMultipleButton
    = new Button("Ouvrir les images...");

    openButton.setOnAction(
        new EventHandler<ActionEvent>() {
            @Passer autre
            public void handle(final ActionEvent e) {
                Fichier file = fileChooser.showOpenDialog(stage); si (fichier != null) {

                    ouvreFichier(fichier);
                }
            }
        });
}

openMultipleButton.setOnAction(
    new EventHandler<ActionEvent>() {
        @Passer autre
        public void handle(final ActionEvent e) {
            Liste<Fichier> liste =
            fileChooser.showOpenMultipleDialog(étape); si (liste != null) {

                for (Fichier fichier : liste) {
                    ouvreFichier(fichier);
                }
            }
        }
    });
}

dernier GridPane inputGridPane = new GridPane();

GridPane.setConstraints(openButton, 0, 0);
GridPane.setConstraints(openMultipleButton, 1, 0); inputGridPane.setHgap(6);
inputGridPane.setVgap(6); inputGridPane.getChildren().addAll(openButton,
openMultipleButton);

rootGroup du dernier volet = new VBox(12);

```

```

rootGroup.getChildren().addAll(inputGridPane);
rootGroup.setPadding(nouveaux inserts(12, 12, 12, 12));

stage.setScene(nouvelle scène(rootGroup));
spectacle();
}

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

private void openFile(Fichier fichier) {
    essayer {
        desktop.open(fichier);
    } catch (IOException ex) {
        Enregistreur.getLogger (
            FileChooserSample.class.getName()).log(
                Niveau.SÉVÈRE, nul, ex
            );
    }
}
}

```

Dans l' [Exemple 26–2](#), le bouton Ouvrir une image permet à l'utilisateur d'ouvrir un sélecteur de fichier pour une seule sélection, et le bouton Ouvrir des images permet à l'utilisateur d'ouvrir un sélecteur de fichier pour plusieurs sélections. Les méthodes setOnAction pour ces boutons sont presque identiques. La seule différence réside dans la méthode utilisée pour appeler un FileChooser.

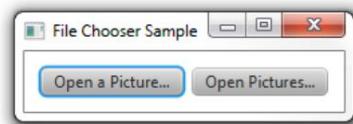
ÿ La méthode showOpenDialog affiche une nouvelle boîte de dialogue d'ouverture de fichier dans laquelle un fichier peut être sélectionné. La méthode renvoie la valeur qui spécifie le fichier choisi par l'utilisateur ou null si aucune sélection n'a été effectuée.

ÿ La méthode showOpenMultipleDialog affiche une nouvelle boîte de dialogue d'ouverture de fichier dans laquelle plusieurs fichiers peuvent être sélectionnés. La méthode renvoie la valeur qui spécifie la liste des fichiers choisis par l'utilisateur ou null si aucune sélection n'a été effectuée. La liste renvoyée ne peut pas être modifiée et lève UnsupportedOperationException à chaque tentative de modification.

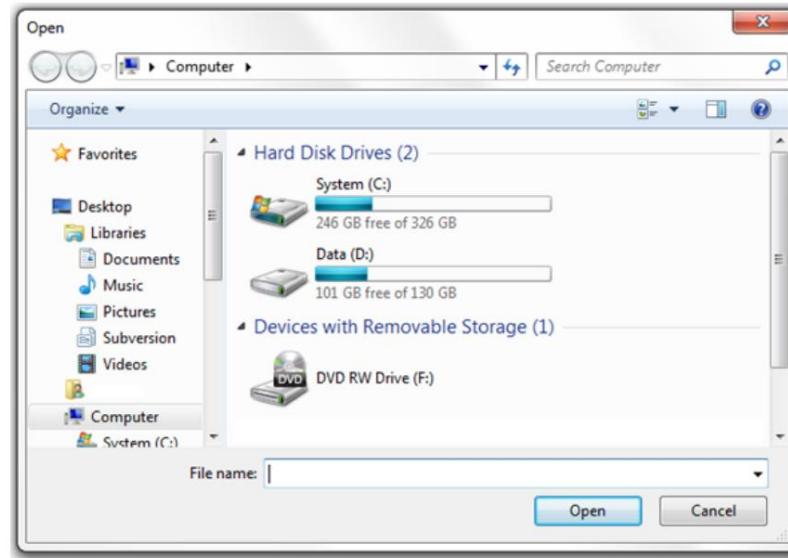
Les deux méthodes ne renvoient pas de résultats tant que la fenêtre de dialogue ouverte affichée n'est pas fermée (en d'autres termes, jusqu'à ce qu'un utilisateur valide ou annule la sélection).

Lorsque vous compilez et exécutez l'application FileChooserSample, elle produit la fenêtre illustrée à la [Figure 26–5](#).

**Figure 26–5 FileChooserSample avec deux boutons**



Lorsque vous cliquez sur l'un des boutons, la fenêtre de dialogue illustrée à la [Figure 26–6](#) apparaît. La fenêtre de dialogue de sélection de fichiers ouverte affiche l'emplacement par défaut de votre système d'exploitation.

**Figure 26–6 Fenêtre de sélection de fichiers par défaut**

Les utilisateurs de l'application `FileChooserSample` peuvent naviguer vers un répertoire contenant des images et sélectionner une image. Une fois qu'un fichier est sélectionné, il est ouvert avec l'application associée. L'exemple de code implémente cela en utilisant la méthode `open` de la classe `java.awt.Desktop` : `desktop.open(file);`.

---

**Remarque**: La disponibilité de la classe `Desktop` dépend de la plate-forme. Reportez-vous à la documentation de l'API pour plus d'informations sur la classe `Desktop`. Vous pouvez également utiliser la méthode `isDesktopSupported()` pour vérifier si elle est prise en charge sur votre système.

---

Vous pouvez améliorer l'expérience utilisateur de cette application en définissant le répertoire du sélecteur de fichiers sur le répertoire spécifique qui contient les images.

## Configuration d'un sélecteur de fichiers

Vous pouvez configurer la fenêtre de dialogue du sélecteur de fichiers en définissant le `initialDirectory` et les propriétés de titre d'un objet `FileChooser`. [L'exemple 26–3](#) montre comment spécifier le répertoire initial et un titre approprié pour prévisualiser et ouvrir les images.

### Exemple 26–3 Définition du répertoire initial et du titre de la fenêtre

```
importer java.awt.Desktop;
importer java.io.File;
import java.io.IOException;
importer java.util.List;
importer java.util.logging.Level;
importer java.util.logging.Logger;
importer javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
```

```
importer javafx.scene.layout.GridPane; importer
javafx.scene.layout.Pane; importer
javafx.scene.layout.VBox; importer
javafx.stage.FileChooser; importer javafx.stage.Stage;

La classe finale publique FileChooserSample étend l'application {

    Bureau privé desktop = Desktop.getDesktop();

    @Passer outre
    début du vide public (étape finale) {
        stage.setTitle("Échantillon de sélecteur de fichiers");

        final FileChooser fileChooser = new FileChooser();

        bouton final openButton = new Button("Ouvrir une image..."); bouton final openMultipleButton
        = new Button("Ouvrir les images...");

        openButton.setOnAction(
            new EventHandler<ActionEvent>() {
                @Passer outre
                public void handle(final ActionEvent e)
                { configureFileChooser(fileChooser); Fichier file =
                    fileChooser.showOpenDialog(stage); si (fichier != null) {

                        ouvreFichier(fichier);
                    }
                }
            });
    });

    openMultipleButton.setOnAction(
        new EventHandler<ActionEvent>() {
            @Passer outre
            public void handle(final ActionEvent e)
            { configureFileChooser(fileChooser);
                Liste<Fichier> liste =
                    fileChooser.showOpenMultipleDialog(étape); si (liste != null) {

                        for (Fichier fichier : liste) {
                            ouvreFichier(fichier);
                        }
                    }
            });
    });

    dernier GridPane inputGridPane = new GridPane();

    GridPane.setConstraints(openButton, 0, 0);
    GridPane.setConstraints(openMultipleButton, 1, 0); inputGridPane.setHgap(6);
    inputGridPane.setVgap(6); inputGridPane.getChildren().addAll(openButton,
    openMultipleButton);

    rootGroup du dernier volet = new VBox(12);
    rootGroup.getChildren().addAll(inputGridPane); rootGroup.setPadding(nouveaux
    inserts(12, 12, 12, 12));

    stage.setScene(nouvelle scène(rootGroup)); spectacle();
```

```
}

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

}

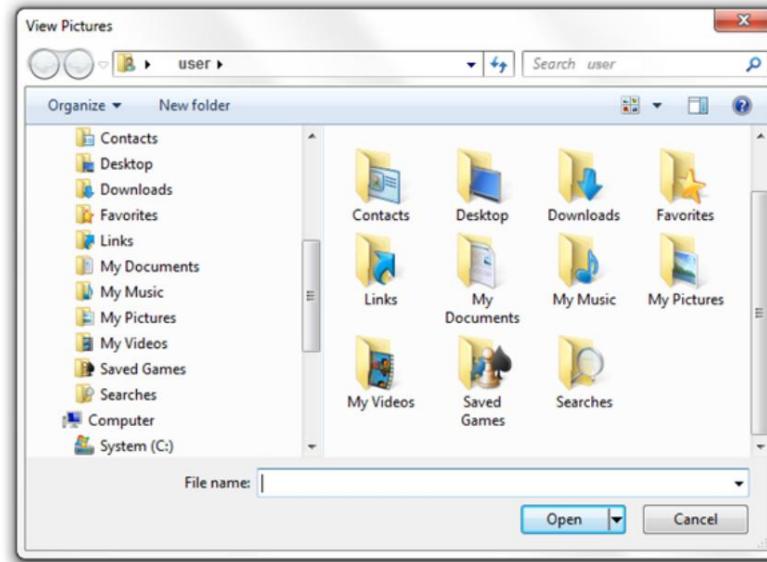
vide statique privé configureFileChooser(fileChooser final FileChooser){
    fileChooser.setTitle("Afficher les images");
    fileChooser.setInitialDirectory( nouveau
        fichier(System.getProperty("user.home"))
    );
}

private void openFile(Fichier fichier) { essayez
    { desktop.open(fichier); } catch (IOException
        ex)
    {

        { Logger.getLogger( FileChooserSample.class.getName()).log( Level.SEVERE,
            null, ex
        );
    }
}
}
```

La méthode `configureFileChooser` définit le titre de la vue `Images` et le chemin d'accès au répertoire de base de l'utilisateur avec le sous-répertoire `Mes images`. Lorsque vous compilez et exécutez `FileChooserSample` et que vous cliquez sur l'un des boutons, le sélecteur de fichier illustré à la Figure 26-7 apparaît.

**Figure 26–7 Ouverture d'une bibliothèque d'images**



Vous pouvez également laisser les utilisateurs spécifier le répertoire cible à l'aide de la classe `DirectoryChooser`. Dans le fragment de code illustré dans l'[Exemple 26-4](#), le clic sur le bouton `seekButton` appelle la méthode `directoryChooser.showDialog`.

**Exemple 26–4 Utilisation de la classe DirectoryChooser**

```
Bouton final BrowseButton = nouveau bouton("...");  
BrowseButton.setOnAction(  
    new EventHandler<ActionEvent>() {  
        @Passer autre  
        public void handle(final ActionEvent e) {  
            DirectoryChooser final DirectoryChooser = new DirectoryChooser();  
            fichier final selectedDirectory =  
            directoryChooser.showDialog(stage); if (selectedDirectory != null) {  
  
                selectedDirectory.getAbsolutePath();  
            }  
        }  
    }  
);
```

Une fois la sélection effectuée, vous pouvez affecter la valeur correspondante au sélecteur de fichier comme suit : fileChooser.setInitialDirectory(selectedDirectory); .

**Définition des filtres d'extension**

Comme option de configuration suivante, vous pouvez définir le filtre d'extension pour déterminer les fichiers à ouvrir dans un sélecteur de fichiers, comme illustré dans l' [Exemple 26–5](#).

**Exemple 26–5 Définition d'un filtre de type d'image**

```
importer java.awt.Desktop; importer  
java.io.File; import java.io.IOException;  
importer java.util.List; importer  
java.util.logging.Level; importer  
java.util.logging.Logger; importer  
javafx.application.Application; import  
javafx.event.ActionEvent; import javafx.event.EventHandler;  
importer javafx.geometry.Insets; import javafx.scene.Scene;  
importer javafx.scene.control.Button; importer  
javafx.scene.layout.GridPane; importer  
javafx.scene.layout.Pane; import javafx.scene.layout.VBox;  
importer javafx.stage.FileChooser; import javafx.stage.Stage;
```

La classe finale publique FileChooserSample étend l'application {

```
Bureau privé desktop = Desktop.getDesktop();  
  
@Passer autre  
début du vide public (étape finale) {  
    stage.setTitle("Échantillon de sélecteur de fichiers");  
  
    final FileChooser fileChooser = new FileChooser(); bouton final openButton =  
    new Button("Ouvrir une image..."); bouton final openMultipleButton = new Button("Ouvrir les  
images...");  
  
    openButton.setOnAction(  
        new EventHandler<ActionEvent>() {
```

```

    @Passer outre
    public void handle(final ActionEvent e)
        { configureFileChooser(fileChooser); Fichier file =
        fileChooser.showOpenDialog(stage); si (fichier != null) {

            ouvreFichier(fichier);
        }
    }
});

openMultipleButton.setOnAction(
    new EventHandler<ActionEvent>() {
        @Passer outre
        public void handle(final ActionEvent e)
            { configureFileChooser(fileChooser);
            Liste<Fichier> liste =
            fileChooser.showOpenMultipleDialog(étape); si (liste != null) {

                for (Fichier fichier : liste) {
                    ouvreFichier(fichier);
                }
            }
        }
    });
}

dernier GridPane inputGridPane = new GridPane();

GridPane.setConstraints(openButton, 0, 1);
GridPane.setConstraints(openMultipleButton, 1, 1); inputGridPane.setHgap(6);
inputGridPane.setVgap(6); inputGridPane.getChildren().addAll(openButton,
openMultipleButton);

rootGroup du dernier volet = new VBox(12);
rootGroup.getChildren().addAll(inputGridPane);
rootGroup.setPadding(nouveaux inserts(12, 12, 12, 12));

stage.setScene(nouvelle scène(rootGroup)); spectacle();

}

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

}

private static void configureFileChooser( final FileChooser
fileChooser) {
    fileChooser.setTitle("Afficher les images");
    fileChooser.setInitialDirectory( nouveau
        fichier(System.getProperty("user.home")))
    );
    fileChooser.getExtensionFilters().addAll( new
        FileChooser.ExtensionFilter("Toutes les images", "*.*"), new
        FileChooser.ExtensionFilter("JPG", "*.jpg"), new
        FileChooser.ExtensionFilter("PNG", ".png")
    );
}

private void openFile(Fichier fichier) {

```

```

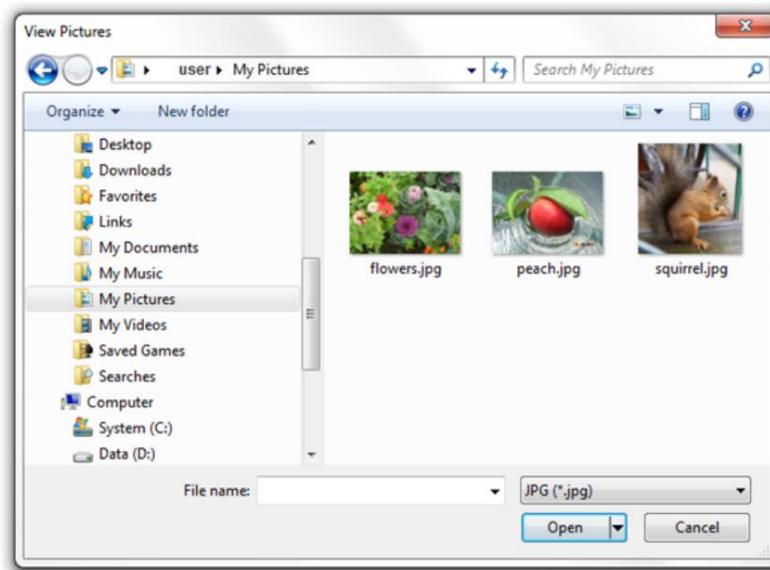
        essayer {
            desktop.open(fichier);
        } catch (IOException ex) {
            Logger.getLogger(FileChooserSample.class.getName()).log(
                Niveau.SÉVÈRE, nul, ex
            );
        }
    }
}

```

Dans l' [Exemple 26–5](#), vous définissez un filtre d'extension à l'aide de FileChooser.ExtensionFilter pour définir les options suivantes pour la sélection de fichiers : Toutes les images, JPG et PNG.

Lorsque vous compilez, exécutez le code FileChooserSample de l' [Exemple 26–5](#) et cliquez sur l'un des boutons, les filtres d'extension apparaissent dans la fenêtre du sélecteur de fichiers. Si un utilisateur sélectionne JPG, le sélecteur de fichiers affiche uniquement les images de type JPG. La [Figure 26–8](#) capture le moment de la sélection des images JPG dans le répertoire Mes images.

**Figure 26–8 Filtrage des fichiers JPG dans le sélecteur de fichiers**



#### Enregistrement de fichiers

En plus d'ouvrir et de filtrer les fichiers, l' API FileChooser permet à un utilisateur de spécifier un nom de fichier (et son emplacement dans le système de fichiers) pour qu'un fichier soit enregistré par l'application. La méthode showSaveDialog de la classe FileChooser ouvre une fenêtre de dialogue d'enregistrement. Comme les autres méthodes de show dialog, la méthode showSaveDialog renvoie le fichier choisi par l'utilisateur ou null si aucune sélection n'a été effectuée.

Le fragment de code illustré dans l' [exemple 26–6](#) est un ajout à l' exemple de [menu](#). Il implémente un élément supplémentaire du menu contextuel qui enregistre l'image affichée dans le système de fichiers.

#### Exemple 26–6 Enregistrement d'une image avec la classe FileChooser

```

MenuItem cmItem2 = new MenuItem("Enregistrer l'image");
cmItem2.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {

```

Enregistrement de fichiers

```

FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Enregistrer l'image");
System.out.println(pic.getId()); Fichier file =
fileChooser.showSaveDialog(stage); si (fichier != null) {

    essayez { ImageIO.write(SwingFXUtils.fromFXImage(pic.getImage(), null), "png",
        fichier); } catch (IOException ex) { System.out.println(ex.getMessage());

    }
}
);

```

Lorsque vous ajoutez l' [Exemple 26–6](#) à l'application MenuSample (trouvez le code source dans les fichiers d'application), que vous le compilez et l'exécutez, vous activez l'élément de menu Enregistrer l'image, comme illustré à la [Figure 26–9](#).

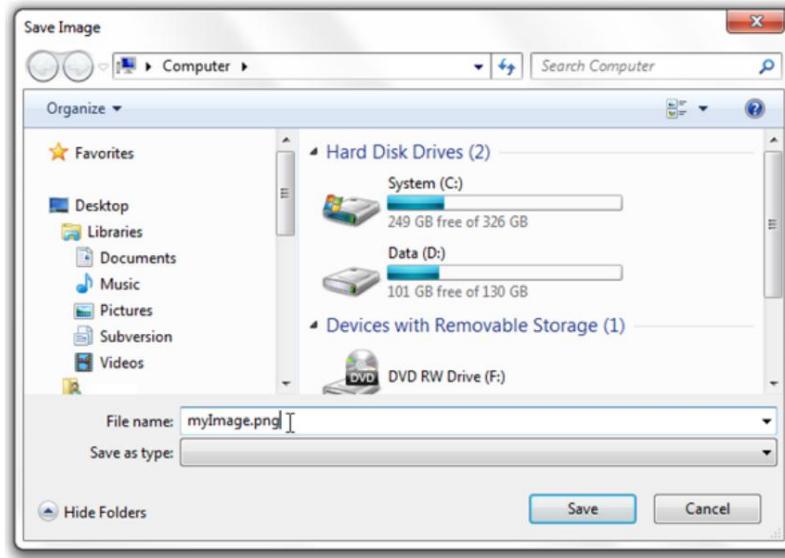
**Figure 26–9** Enregistrement de l'image



The hawthorn is a large genus of shrubs and trees in the rose family, Rosaceae, native to temperate regions of the Northern Hemisphere in Europe, Asia and North America. The name hawthorn was originally applied to the species native to northern Europe, especially the Common Hawthorn C. monogyna, and the unmodified name is often so used in Britain and Ireland.

Lorsqu'un utilisateur sélectionne l'élément Enregistrer l'image, la fenêtre Enregistrer l'image illustrée à la [Figure 26–10](#) s'affiche.

Figure 26–10 La fenêtre Enregistrer l'image



La fenêtre Enregistrer l'image correspond à l'expérience utilisateur typique pour les fenêtres de dialogue d'enregistrement: l'utilisateur doit sélectionner le répertoire cible, saisir le nom du fichier d'enregistrement et cliquer sur Enregistrer.

#### Documentation relative à l'API

ÿ Sélecteur de fichiers

ÿ DirectoryChooser

Enregistrement de fichiers

---

## 27Personnalisation des commandes de l'interface utilisateur

Ce chapitre décrit les aspects de la personnalisation des contrôles de l'interface utilisateur et résume certains conseils et astuces fournis par Oracle pour vous aider à modifier l'apparence et le comportement des contrôles de l'interface utilisateur.

Vous pouvez apprendre à personnaliser les contrôles à partir des exemples d'applications du projet UIControlSamples en appliquant des feuilles de style en cascade (CSS), en redéfinissant le comportement par défaut et en utilisant des fabriques de cellules. Pour des cas plus spécifiques, lorsque la tâche de votre application nécessite des fonctionnalités uniques qui ne peuvent pas être implémentées avec les classes du package javafx.scene.control, étendez la classe Control pour inventer votre propre contrôle.

### Appliquer CSS

Vous pouvez modifier l'apparence des contrôles de l'interface utilisateur en redéfinissant les définitions de style des feuilles de style caspien JavaFX. Skinning JavaFX Applications with CSS explique les concepts généraux et les approches pour modifier les styles et les activer dans une application JavaFX.

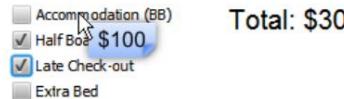
Considérez certaines des tâches spécifiques fréquemment demandées par les développeurs sur le forum JavaFX.

Bien que la classe Tooltip n'ait pas de propriétés ou de méthodes pour changer la couleur par défaut de l'info-bulle, vous pouvez modifier la propriété -fx-background-color de la classe CSS .tooltip comme illustré dans l'[Exemple 27-1](#).

#### Exemple 27-1 Modification de la couleur d'arrière-plan d'une info-bulle

```
.info-bulle {
    -fx-background-color: linear-gradient(#e2ecfe, #99bcfd);
}
.page-coin {
    -fx-background-color: dégradé linéaire (de 0% 0% à 50% 50%, #3278fa, #99bcfd);
}
```

La classe CSS .page-corner définit la couleur du coin inférieur droit de l'info-bulle. Lorsque vous ajoutez le code de l'[Exemple 27-1](#) aux feuilles de style de TooltipSample et appliquez les feuilles de style à la scène, l'info-bulle change de couleur en bleu. Voir [Figure 27-1](#) pour évaluer l'effet.

**Figure 27–1 Info-bulle avec la couleur d'arrière-plan bleue**

Notez que lorsque vous modifiez le style par défaut d'une info-bulle, le nouveau look est appliqué à toutes les info-bulles de votre application.

Une autre tâche de conception courante consiste à modifier les marques par défaut des contrôles. Par exemple, le style par défaut de la classe CheckBox définit la coche traditionnelle pour l'état sélectionné. Vous pouvez redéfinir la forme du repère ainsi que sa couleur comme illustré dans l'[exemple 27–2](#).

#### **Exemple 27–2 Marque alternative pour une case à cocher**

```
.case à cocher .marque {
    -fx-forme: ;
    "M2,0L5,4L8,0L10,0L10,2L6,5L10,8L10,10L8,10L5,6L2,10L0,10L0,8L4,5L0,2L0,0Z";
}
.case à cocher: sélectionné .mark {
    -fx-background-color: #0181e2;
}
```

La propriété `-fx-shape` définit le nouveau chemin SVG pour la marque et la propriété `-fx-background-color` définit sa couleur. Lorsque les feuilles de style modifiées sont activées dans l'application CheckBoxSample, les cases à cocher sélectionnées contiennent des marques X au lieu de coches, comme illustré à la [Figure 27–2](#).

**Figure 27–2 ComboBoxSample avec le style de case à cocher modifié**

De nombreux développeurs ont demandé comment surmonter la limitation du style visuel des contrôles TableView et ListView. Par défaut, toutes les lignes de ces contrôles sont affichées, qu'elles soient vides ou non. Avec les paramètres CSS appropriés, vous pouvez définir une couleur spécifique pour toutes les lignes vides. [L'exemple 27–3](#) implémente cette tâche pour un contrôle TableView.

#### **Exemple 27–3 Définition de la couleur des lignes vides dans une vue de tableau**

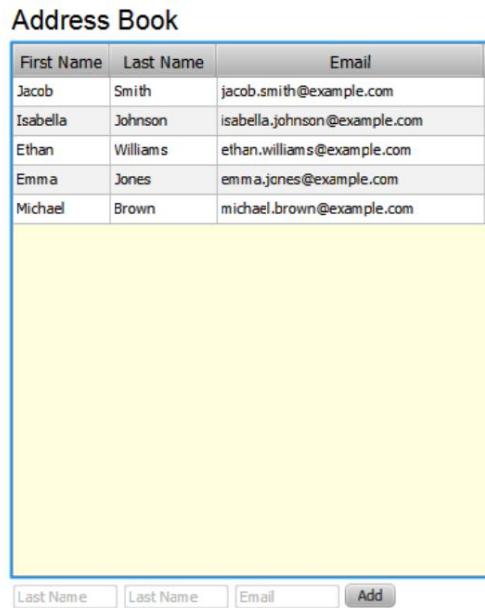
```
.table-row-cell:empty {
    -fx-background-color: jaune clair;
}

.table-row-cell:vide .table-cell {
    -fx-bordure-largeur: 0px;
}
```

Le premier style CSS détermine que toutes les lignes vides, qu'elles soient paires ou impaires, doivent avoir un arrière-plan jaune clair. Lorsque table-row-cell est vide, la deuxième instruction CSS supprime la bordure verticale qui est peinte sur le côté droit de toutes les cellules du tableau.

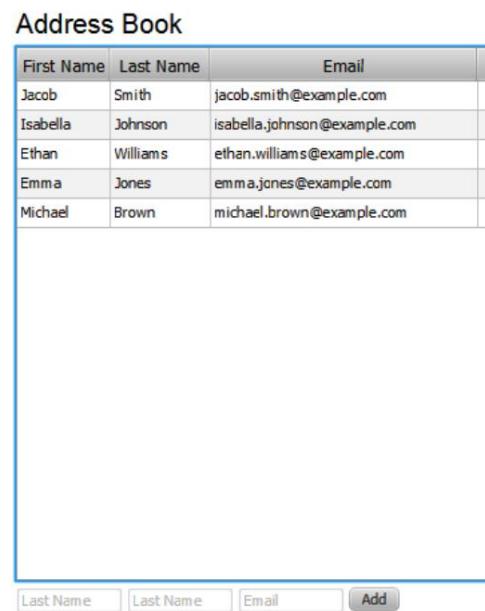
Lorsque les styles CSS de l' [Exemple 27-3](#) sont activés dans l'application TableViewSample, la table Carnet d'adresses ressemble à celle illustrée à la [Figure 27-3](#).

**Figure 27–3** TableViewSample avec couleur ajoutée aux lignes vides



Vous pouvez même définir la valeur nulle pour la couleur d'arrière-plan des cellules vides. Les feuilles de style utiliseront la couleur d'arrière-plan par défaut de la vue du tableau dans ce cas. Voir Figure 27-4 pour évaluer l'effet.

**Figure 27–4 TableViewSample avec une couleur d'arrière-plan nulle ajoutée aux lignes vides**



Vous pouvez définir davantage de propriétés CSS pour les contrôles d'interface utilisateur afin de modifier leurs formes, leurs schémas de couleurs et les effets appliqués. Consultez le guide de référence JavaFX CSS pour plus d'informations sur les propriétés et les classes CSS disponibles.

### Modification du comportement par défaut

De nombreux développeurs ont demandé une API spécifique pour restreindre la saisie dans le champ de texte, par exemple, pour n'autoriser que les valeurs numériques. [L'exemple 27–4](#) fournit une application simple avec un champ de texte numérique.

### Exemple 27–4 Interdire les lettres dans le champ de texte

```
importer javafx.application.Application; import
javafx.event.ActionEvent; import javafx.event.EventHandler;
importer javafx.geometry.Insets; importer javafx.scene.Group;
importer javafx.scene.Scene; importer javafx.scene.control.*;
importer javafx.scene.layout.GridPane; import
javafx.scene.layout.HBox; importer javafx.stage.Stage;

la classe publique CustomTextFieldSample étend l'application {

    Étiquette statique finale étiquette = nouvelle étiquette ();

    @Passer outre
    public void start(Stage stage) {
        Racine de groupe = nouveau groupe ();
        Scene scene = new Scene(root, 300, 150);
        stage.setScene(scene);
        stage.setTitle("Exemple de champ de texte");

        GridPane grid = new GridPane();
        grid.setPadding(nouveaux inserts(10, 10, 10, 10)); grille.setVgap(5);
        grille.setHgap(5);

        scene.setRoot(grille); dollar final
        de l'étiquette = nouvelle étiquette ("$");
        GridPane.setConstraints(dollar, 0, 0); grille.getChildren().add(dollar);

        somme finale de TextField = new TextField()
        @Passer outre
        public void replaceText(int start, int end, String text) { if (!text.matches("[az, AZ]")) {

            super.replaceText(début, fin, texte);
        }
        label.setText("Entrez une valeur numérique");
    }

    @Passer outre
    public void replaceSelection(String text) { if (!text.matches("[az,
    AZ]")) { super.replaceSelection(text);

    }
}
```

```

});  
  

sum.setPromptText("Entrez le total");  

sum.setPrefColumnCount(10);  

GridPane.setConstraints(somme, 1, 0);  

grille.getChildren().add(somme);  
  

Button submit = new Button("Soumettre");  

GridPane.setConstraints(soumettre, 2, 0);  

grille.getChildren().add(soumettre);  
  

submit.setOnAction(new EventHandler<ActionEvent>() {  

    @Passer outre  

    public void handle(ActionEvent e) { label.setText(null);  

    }  
});  
  

GridPane.setConstraints(étiquette, 0, 1);  

GridPane.setColumnSpan(étiquette, 3);  

grille.getChildren().add(étiquette);  
  

scene.setRoot(grille);  

spectacle();  
}  
  

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

}  
}

```

Pour redéfinir l'implémentation par défaut de la classe `TextField`, vous devez remplacer les méthodes `replaceText` et `replaceSelection` héritées de la classe `TextInputControl`.

Lorsque l'utilisateur essaie d'entrer une lettre dans le champ de texte Somme, aucun symbole n'apparaît et le message d'avertissement s'affiche. [La figure 27–5](#) illustre cette situation.

**Figure 27–5 Tentative de saisie de symboles alphabétiques**



Cependant, lorsque l'utilisateur tente d'entrer des valeurs numériques, elles apparaissent dans le champ comme illustré à la [Figure 27–6](#).

**Figure 27–6 Saisie de valeurs numériques**



## Mise en œuvre des usines cellulaires

L'apparence et même le comportement de quatre contrôles de l'interface utilisateur peuvent être entièrement personnalisés en utilisant le mécanisme des usines de cellules. Vous pouvez appliquer des fabriques de cellules à TableView, ListView, TreeView et ComboBox. Une fabrique de cellules est utilisée pour générer des instances de cellule, qui sont utilisées pour représenter n'importe quel élément de ces contrôles.

La classe Cell étend la classe Labeled, qui fournit toutes les propriétés et méthodes requises pour implémenter le cas d'utilisation le plus courant : afficher et modifier du texte.

Cependant, lorsque la tâche de votre application nécessite d'afficher des objets graphiques dans les listes ou les tableaux, vous pouvez utiliser la propriété graphique et placer n'importe quel nœud dans la cellule (voir la spécification de l'API de classe Cellule pour plus d'informations sur les cellules personnalisées).

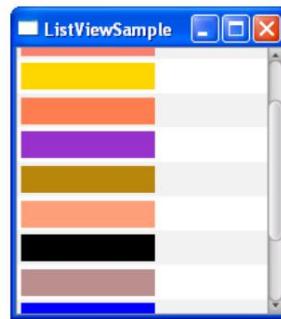
Par exemple, les fragments de code de l' [exemple 27–5](#) créent une fabrique de cellules pour la vue de liste et redéfinissent le contenu des cellules dans la méthode updateItem, de sorte que la liste affiche des rectangles de différentes couleurs.

### Exemple 27–5 Implémentation de fabriques de cellules pour le contrôle ListView

```
list.setCellFactory(new Callback<ListView<String>, ListCell<String>>() {
    Remplacer public ListCell<String> call(ListView<String> list) {
        return new ColorRectCell();
    }
});
...
la classe statique ColorRectCell étend ListCell<String> {
    @Passer autre
    public void updateItem (élément de chaîne, booléen vide) {
        super.updateItem(élément, vide);
        Rectangle rect = nouveau Rectangle(100, 20);
        si (élément != null) {
            rect.setFill(Color.web(item));
            setGraphic(rect);
        } autre {
            setGraphic(null);
        }
    }
}
```

[La Figure 27–7](#) montre à quoi ressemble cette liste personnalisée dans le ListViewSample du projet UIControlSamples.

**Figure 27–7 Affichage en liste avec rectangles de couleur**



Ce didacticiel utilise largement le mécanisme de fabrique de cellules pour personnaliser les contrôles de l'interface utilisateur.

[Le Tableau 27–1](#) récapitule les modèles de codage que vous pouvez utiliser pour implémenter des fabriques de cellules sur vos applications.

**Tableau 27–1 Modèles de codage d'usine de cellules**

Contrôleur	Modèle de codage
ListView, Boîte combo	<pre>list.setCellFactory(nouveau rappel&lt;ListView&lt;String&gt;, ListCell&lt;String&gt;&gt;() { @Override      public ListCell&lt;String&gt; call(ListView&lt;String&gt; list) {         // implémentation de la cellule     } });  TableView column.setCellFactory(nouveau rappel&lt;TableColumn, TableCell&gt;() {     public TableCell call(TableColumn p) {         // implémentation de la cellule     } });  TreeView tree.setCellFactory(new Callback&lt;TreeView&lt;String&gt;, TreeCell&lt;String&gt;&gt;(){      @Passer autre     public TreeCell&lt;String&gt; call(TreeView&lt;String&gt; p) {         // implémentation de la cellule     } });</pre>

Vous pouvez personnaliser ces contrôles à l'aide du mécanisme de fabrique de cellules ou utiliser les implémentations préfabriquées de l'éditeur de cellules qui fournissent des modèles de données spécifiques sous-jacents à la visualisation. Le [Tableau 27–2](#) répertorie les classes correspondantes disponibles dans l'API JavaFX.

**Tableau 27–2 Classes de l'Éditeur de cellules pour les contrôles de vue de liste, d'arborescence et de vue de tableau**

Contrôleur	Classes de l'éditeur de cellule
Vue liste	<ul style="list-style-type: none"> <li>ÿ CheckBoxListCell</li> <li>ÿ ChoiceBoxListCell</li> <li>ÿ ComboBoxListCell</li> <li>ÿ TextFieldListCell</li> </ul>
Arborescence	<ul style="list-style-type: none"> <li>ÿ CheckBoxTreeCell</li> <li>ÿ ChoiceBoxTreeCell</li> <li>ÿ ComboBoxTreeCell</li> <li>ÿ TextFieldTreeCell</li> </ul>
Vue de tableau	<ul style="list-style-type: none"> <li>ÿ CheckBoxTableCell</li> <li>ÿ ChoiceBoxTableCell</li> <li>ÿ ComboBoxTableCell</li> <li>ÿ ProgressBarTableCell</li> <li>ÿ TextFieldTableCell</li> </ul>

Chaque classe d'éditeur de cellule dessine un nœud spécifique à l'intérieur de la cellule. Par exemple, la classe CheckBoxListCell dessine un nœud CheckBox à l'intérieur de la cellule de liste.

Pour évaluer d'autres cas d'utilisation de fabrique de cellules et d'éditeur de cellules, consultez les chapitres [Table View](#), [Tree View](#) et [Combo Box](#).

#### Documentation et ressources connexes

- ÿ Applications d'habillage avec CSS
- ÿ Guide de référence JavaFX CSS
- ÿ Actualités, démos et aperçu JavaFX