Cours IHM-1
JavaFX

8 - Menus
Choix / Sélection

Menus déroulants
Menus contextuels
Mnémoniques / Accélérateurs

Menu [1]

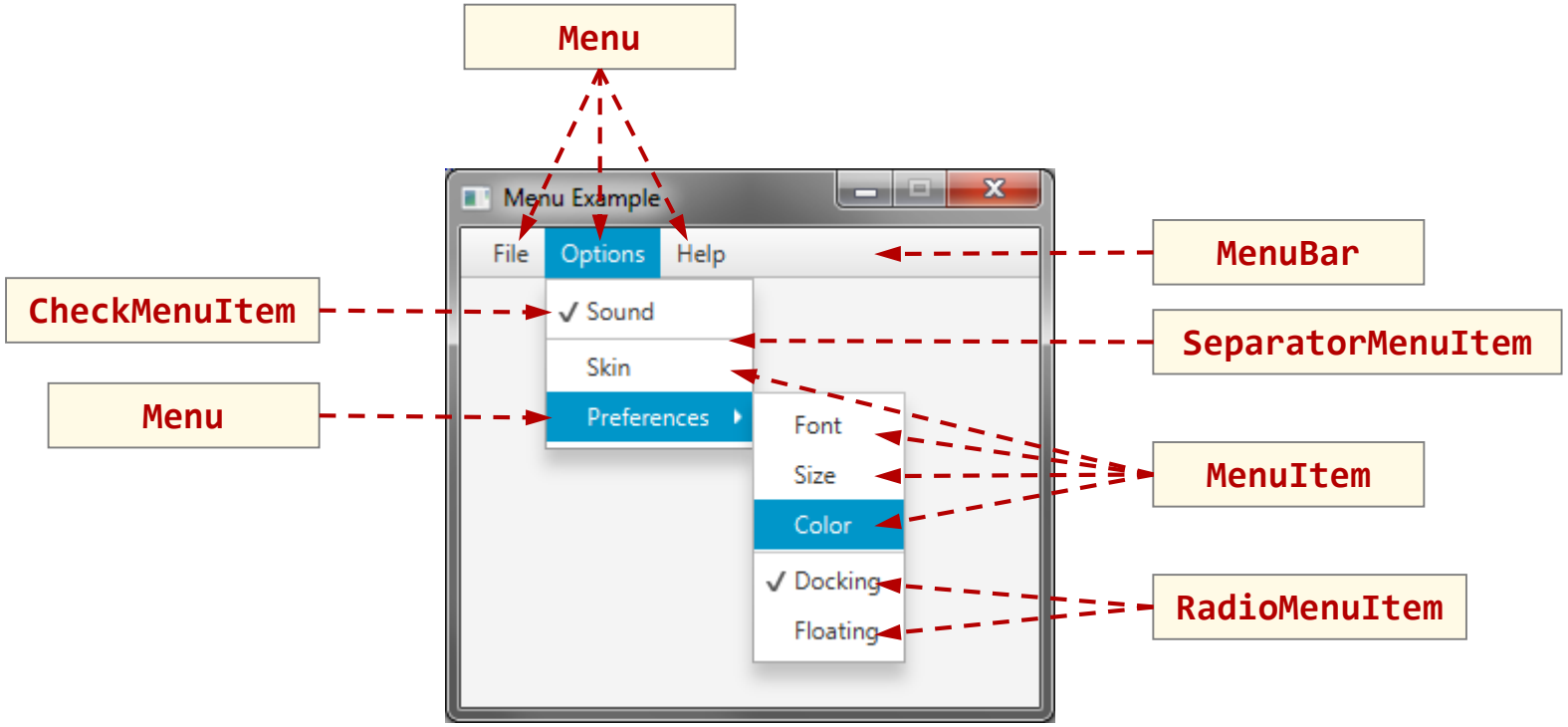
- Les **menus** sont des éléments de l'interface permettant à l'utilisateur de choisir des options qui pourront déclencher des actions et/ou changer l'état de certaines propriétés de l'application.
- Le principe du menu est (comme au restaurant) que l'utilisateur puisse voir et parcourir la liste des options avant de se décider.
- Dans les applications, les menus peuvent prendre différentes formes. Parmi les plus classiques, on trouve :
  - Les **menu déroulants** *(drop-down menu)*
    - ⇨ Des en-têtes de menus sont placés dans un conteneur sous forme de barre
    - ⇨ Un clic sur ces en-têtes "déroule" le menu et fait apparaître, dans une fenêtre *popup*, les options de ce menu.
  - Les **menu contextuels** *(popup menu)*
    - ⇨ Menus affichés en réaction à un événement, généralement une action de la souris (clic-droit), l'activation d'une touche ou un geste.
    - ⇨ Les options du menu qui sont affichées dépendent de l'endroit où l'on a cliqué (contexte).

Menu [2]

- Les options des menus peuvent elles-mêmes ouvrir d'autres menus. On parle alors de *sous-menus* qui peuvent s'ouvrir en cascade (on peut avoir plusieurs niveaux de sous-menus).
- Dans la librairie *JavaFX*, un certain nombre de composants sont dédiés aux menus et sont utilisés pour les construire :
  - `MenuBar`
  - `Menu`
  - `MenuItem`
  - `CheckMenuItem`
  - `RadioMenuItem`
  - `CustomMenuItem`
  - `SeparatorMenuItem`
  - `Contextmenu`

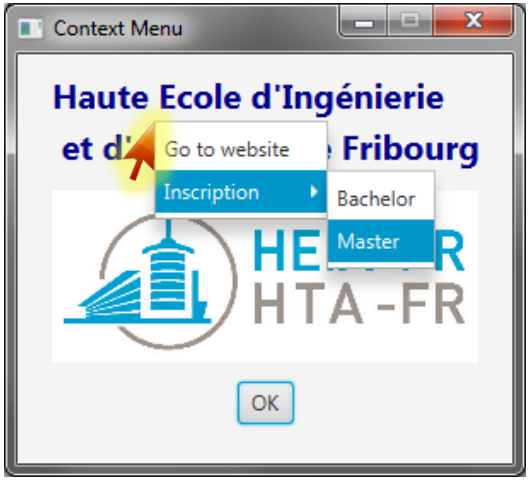
Menu [3]

- Les **différents composants** qui interviennent dans la gestion des **menus déroulants** sont illustrés dans l'exemple suivant :



Menu [4]

- Pour les **menus contextuels**, ce sont pratiquement les mêmes composants qui sont utilisés à la simple différence que :
  - Le composant `MenuBar` n'est plus nécessaire.
  - A la place, c'est le composant **`ContextMenu`** qui est utilisé pour rassembler les différentes options du menu contextuel.
- L'affichage du composant sera déclenché par une action de l'utilisateur (en général un clic-droit sur un conteneur ou composant).
- Un menu contextuel peut également contenir des sous-menus en cascade.



Barre de menu [1]

- La barre de menus, représentée par le composant **`MenuBar`**, est un conteneur permettant de rassembler les en-têtes des menus.
- Le composant `MenuBar` doit être placé dans un conteneur de l'interface (par exemple en haut d'un `BorderPane`).
- La création d'une barre de menus s'effectue à l'aide des instructions suivantes :
  - **Création** de la barre de menus :
    ```
    MenuBar mBar = new MenuBar();
    ```
  - L'**ajout des menus** (composants de type `Menu`) dans la barre peut se faire de manière individuelle (en séquence) ou collective :
    ```
    mBar.getMenus().add(mnuFile);
    mBar.getMenus().addAll(mnuFile, mnuOptions, mnuHelp);
    ```
- La propriété `useSystemMenuBar` permet d'indiquer que le mécanisme standard de menu de la plateforme cible doit être utilisé (par exemple sur *MacOS*).

Menu déroulant [1]

- Le composant **Menu** représente un conteneur d'éléments de menu (options) et peut se présenter "enroulé" (en-tête) ou "déroulé" (fenêtre *popup*) lorsqu'on clique dessus.
- Il peut être ajouté à une barre de menus (**MenuBar**) ou être inséré comme sous-menu dans un autre composant de type **Menu**.
- Lorsque le composant **Menu** représente un sous-menu, il est accompagné d'un indicateur visuel (généralement une flèche dirigée vers la droite) qui le distingue d'un élément terminal (option de menu).
- Création et alimentation d'un objet de type **Menu** :
  - **Création** d'un menu :
    ```
    Menu mnuFile = new Menu("File");
    ```
  - L'**ajout des options de menus** (`MenuItem`, `CheckMenuItem`, …) dans le menu peut se faire de manière individuelle (en séquence) ou collective :
    ```
    mnuFile.getItems().add(mniSave);
    mnuFile.getItems().addAll(mniSave, mniSaveAs, mniQuit);
    ```

Menu déroulant [2]

- Un sous-menu est donc simplement créé en ajoutant un composant de type `Menu` dans un autre composant du même type :
  - **Création** d'un menu et d'un sous-menu :

```
Menu mnuOptions = new Menu("Options");
Menu mnuPrefs   = new Menu("Preferences");
```

  - L'**ajout du menu comme sous-menu** :

```
mnuOptions.getItems().add(mnuPrefs);
```

- Le nombre de niveaux d'imbrication des sous-menus n'est pas limité mais on ne dépasse généralement pas trois niveaux (des exceptions sont possibles).

Options de menu [1]

- Un menu déroulant ou contextuel peut contenir différents éléments (appelés *options de menu*) permettant à l'utilisateur de déclencher une action ou d'activer/désactiver l'élément.
- L'option de base est représentée par le composant **`MenuItem`** qui agit, du point de vue de l'utilisateur, comme un bouton.
- Lors de la création d'une option de menu, on peut lui assigner un texte et optionnellement un composant complémentaire *(graphic)* qui ne devrait pas dépasser 16x16 pixels.
  - **Création** d'une option de menu :

```
MenuItem   mniSave    = new MenuItem("Save");
ImageView  image      = new ImageView(. . .);
MenuItem   mniPrefs   = new MenuItem("Preferences", image);
```

  - **Ajout d'un graphique** :

```
mniSave.setGraphic(new Circle(0, 0, 5, Color.RED));
```

Options de menu [2]

- Pour le **traitement de l'action** déclenchée par le composant `MenuItem` on peut utiliser la méthode utilitaire **`setOnAction(…)`**.
- Exemple :

```
. . .
//--- Traitement de l'action des options de menu
mniQuit.setOnAction(event -> {
    Platform.exit();
  });

mniAbout.setOnAction(event -> {
    System.out.println("About MenuItem Activated");
  });
. . .
```

Options de menu [3]

- Comme option de menu, on peut aussi utiliser le composant **`CheckMenuItem`** qui agit comme une case à cocher placée dans un menu déroulant ou contextuel (l'aspect visuel dépend du thème).
- La propriété `selected` indique si l'option est sélectionnée ou non.
- La propriété `graphic` permet d'ajouter au libellé un composant complémentaire (généralement un graphique de petite taille).
  - **Création** d'une option de menu de type *case à cocher* :

```
CheckMenuItem  cmiSound  = new CheckMenuItem("Sound");
ImageView      image     = new ImageView(. . .);
CheckMenuItem  cmiNotify = new CheckMenuItem("Notify", image);
```

  - **Ajout d'un graphique** :

```
cmiSound.setGraphic(new Rectangle(8, 8, Color.GREEN));
```

Options de menu [4]

- Comme pour le composant `MenuItem`, on peut **traiter l'événement** du composant `CheckMenuItem` en invoquant la méthode utilitaire `setOnAction(…)`.
- Parfois, on ne réagit pas directement au clic sur une telle option mais on prend en compte l'état (sélectionné ou non-sélectionné) dans le traitement d'autres événements (ceux déclenchés par des boutons ou des options de menu de type `MenuItem` par exemple).
- Il est également possible de lier *(binding)* la propriété `selected` de ce composant à une propriété du modèle. Ainsi, le modèle de l'application reflète de manière synchrone l'état de cette propriété sans qu'il soit nécessaire de gérer des événements.

Options de menu [5]

- Le composant **`RadioMenuItem`** qui est une sous-classe de `MenuItem` peut également être utilisé comme option dans un menu. Il agit comme un bouton radio placé dans un menu déroulant et permet la sélection d'une option parmi plusieurs.
- Pour avoir ce comportement de sélection mutuellement exclusive, le composant doit être placé dans un `ToggleGroup` (comme c'est le cas pour le `RadioButton`).
- La propriété `selected` indique si l'option est sélectionnée ou non.
- La propriété `graphic` permet d'ajouter au libellé un composant complémentaire (généralement un graphique de petite taille).
  - **Création** d'une option de menu de type *radio* :

```
RadioMenuItem  rmiDock  = new RadioMenuItem("Docking");
ImageView      image    = new ImageView(. . .);
RadioMenuItem  rmiFloat = new RadioMenuItem("Floating", image);
```

Options de menu [6]

- Le composant **RadioMenuItem** ne devrait jamais être utilisé seul mais faire partie d'un groupe (comme pour les boutons radios).
- Toutes les options qui font partie du même groupe doivent être placées dans un objet commun de type ToggleGroup (toggleGroup est une propriété du composant RadioMenuItem).

```
ToggleGroup tgGroup = new ToggleGroup();
rmiDock.setToggleGroup(tgGroup);
rmiFloat.setToggleGroup(tgGroup);
```

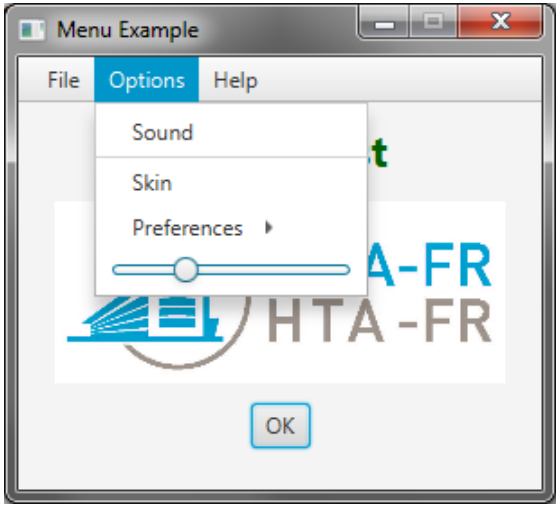
- Pour le **traitement des événements**, toutes les indications et remarques formulées pour le composant CheckMenuItem (voir pages précédentes) s'appliquent aussi au composant RadioMenuItem.

Options de menu [7]

- Le composant **CustomMenuItem** permet de traiter n'importe quel nœud (Node) d'un graphe de scène comme un élément de menu.
- Exemple d'un curseur *(Slider)* placé dans un menu :

```
Slider slrSpeed = new Slider(0, 100, 30);
CustomMenuItem cmiSlider = new CustomMenuItem(slrSpeed);
cmiSlider.setHideOnClick(false);
mnuOptions.getItems().add(cmiSlider);
```

- La propriété hideOnClick permet d'indiquer si le composant et toutes les autres options visibles doivent être masquées lorsqu'on clique sur cet élément (`true` par défaut).
- Les événements du composant inséré dans le menu doivent naturellement être gérés.



Options de menu [8]

- Il est possible de grouper visuellement les options de menu en séparant les différents groupes d'options par une ligne horizontale.
- Le composant **`SeparatorMenuItem`** (qui est une sous-classe de `CustomMenuItem`) permet de créer un tel séparateur.
- Pour chaque séparateur, il faut créer un nouvel objet (on ne peut pas ajouter plusieurs fois un même séparateur).

```
private SeparatorMenuItem sep1     = new SeparatorMenuItem();
private SeparatorMenuItem sep1     = new SeparatorMenuItem();
mnuPrefs.getItems().addAll(mniFont, mniSize,  sep1,
                           rmiDock, rmiFloat, sep2,
                           . . .                    );
```

Menu contextuel [1]

- Les menus contextuels *(popup menu)* se construisent de la même manière que les menus déroulants mais les différents options du menu sont assemblées dans un composant **`ContextMenu`** (au lieu de créer des en-têtes de menus et de les ajouter dans un `MenuBar`).
- Instructions pour gérer un menu contextuel :
  - **Création** du menu contextuel :

```
ContextMenu ctxMenuIm   = new ContextMenu();
```

  - L'**ajout des options de menus** peut se faire de manière individuelle (en séquence) ou collective :

```
ctxMenuIm.getItems().add(mniWeb);
ctxMenuIm.getItems().addAll(mniCopy, mniSave, mniResize);
```

- Des **sous-menus** peuvent également être créés dans un menu contextuel en imbriquant des composants de type `Menu` (de la même manière que pour les menus déroulants).

Menu contextuel [2]

- L'association des menus contextuels aux composants peut se faire de deux manières.
  - Pour les composants qui sont des **sous-classes de `Control`**, le plus simple est d'utiliser la propriété `contextMenu`.

```
ContextMenu ctxMenuIm = new ContextMenu();
ctxMenuIm.getItems().addAll(mniCopy, mniSave, mniResize);
lblImage.setContextMenu(ctxMenuIm);
```

  - Pour les **autres composants** et notamment les **conteneur**s (sous-classes de `Pane`), il faut gérer l'événement provoqué par le clic-droit de la souris et utiliser la méthode `show()` pour afficher le menu (à la position du clic).

```
root.setOnMousePressed(new EventHandler<MouseEvent>() {
  @Override
  public void handle(MouseEvent ev) {
    if (ev.isSecondaryButtonDown()) {
      ctxMenuIm.show(root, ev.getScreenX(), ev.getScreenY());
    }
  }
});
```

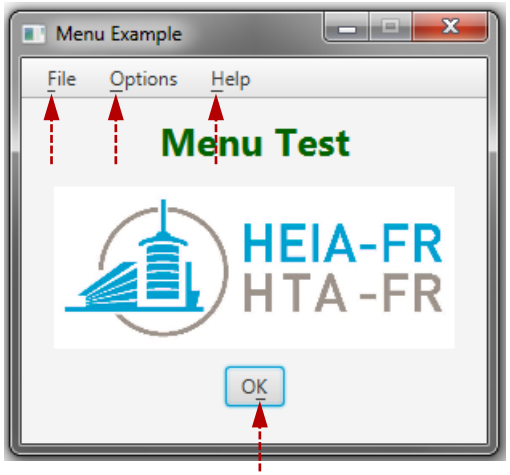
Raccourcis clavier

- Il est souvent souhaitable que certaines actions que l'on peut déclencher avec la souris puissent être également déclenchées à l'aide du clavier.
- C'est le rôle des **raccourcis clavier** qui existent sous deux formes principales :
  - Les **mnémoniques**
    - ⇨ Activation du composant en pressant sur une touche spécifique qui dépend de la plateforme (touche *Alt* sur *Windows*) associée à un autre caractère (généralement alphanumérique).
    - ⇨ Les mnémoniques sont associés à des composants qui possède un libellé et le caractère associé au mnémonique est généralement souligné.
  - Les **accélérateurs**
    - ⇨ Activation de l'action associée à une option de menu par une combinaison unique de touches du clavier.

Mnémonique

- Des **mnémoniques** peuvent être associés à tous les composants possédant un libellé (sous-classes de `Labeled`) ainsi qu'aux menus et aux différentes options de menus.
- Pour ces composants, la propriété booléenne `mnemonicParsing` indique si le **caractère souligné** `'_'` **dans le libellé** du composant doit être considéré comme l'annonce d'un mnémonique pour le caractère suivant.

```
Menu   mnuFile    = new Menu("_File");
Menu   mnuOptions = new Menu("_Options");
Button btnOk      = new Button("O_K");
```



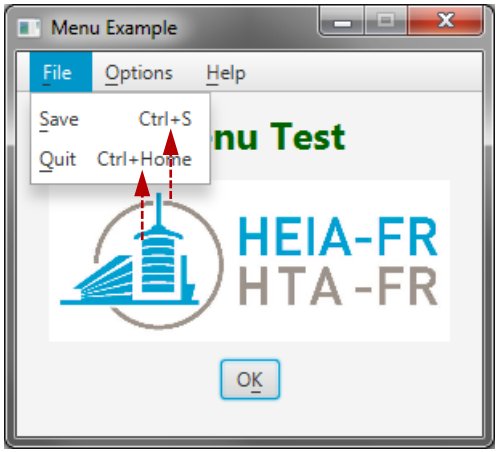
  - La propriété `mnemonicParsing` est à `true` par défaut pour la plupart des composants.

Accélérateur [1]

- Des **accélérateurs** peuvent être associés à tous les composants de type `Menu` (c'est rarement utile) et aux différentes options de menus `MenuItem`, `CheckMenuItem`, …
- Pour ces composants, la propriété `accelerator`, qui est de type `KeyCombination` indique la combinaison de touches qui permet de déclencher l'action associée.
- Il existe différentes manières de créer des objets de type `KeyCombination`. Une des plus simples est d'utiliser la méthode statique `keyCombination("`*key-pattern*`")` qui prend en paramètre une chaîne de caractères qui sera interprétée.
  - Cette chaîne de caractères comprendra un ou plusieurs *modificateurs* et un caractère séparés par le symbole plus '**+**'
  - Modificateurs possibles : `Shift`, `Ctrl`, `Alt`, `Meta`, `Shortcut`
  - Exemples : `"Shift+A"`, `"Ctrl+S"`, `"Alt+Ctrl+M"`

Accélérateur [2]

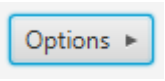
- Pour les touches spéciales (*Enter*, *Home*, *Page_Up*, …), la classe `KeyCode` définit des constantes énumérées représentant les différentes touches du clavier. Le nom de ces constantes peut généralement être utilisé dans la chaîne de caractères passée à la méthode `keyCombination()`.
- La classe `KeyCodeCombination` peut également être utilisée pour créer des combinaisons de touches (voir exemple).
- Exemples de création d'accélérateurs sur des options de menu :



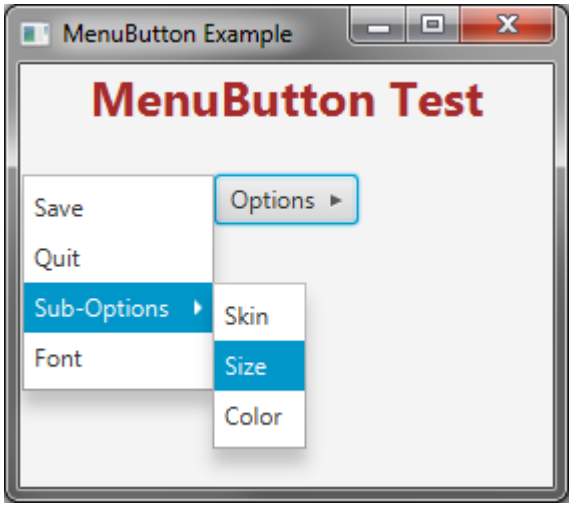

```
mniSave.setAccelerator(KeyCombination.keyCombination("Ctrl+S"));
mniAbout.setAccelerator(KeyCombination.keyCombination("Shift+Ctrl+A"));
mniQuit.setAccelerator(KeyCombination.keyCombination("Ctrl+Home"));
mnuOpt.setAccelerator(KeyCombination.keyCombination("Alt+Shift+Ctrl+O"));
mniSize.setAccelerator(new KeyCodeCombination(
            KeyCode.S, KeyCombination.ALT_DOWN, KeyCombination.SHIFT_DOWN));
```

MenuButton / SplitMenuButton
ChoiceBox / ComboBox
ListView / Spinner

MenuButton



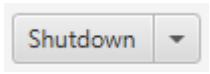
- Le composant **`MenuButton`** est un bouton qui affiche un menu *popup* lorsqu'il est cliqué (à la manière d'un menu contextuel).
- C'est une sous-classe de `ButtonBase` et son API est très proche de celle du composant `Menu`.
- La propriété `popupSide` permet de définir de quel côté doit s'ouvrir le menu *popup*.



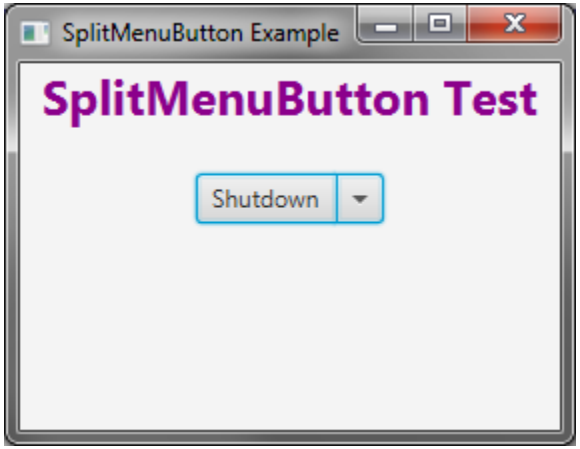
- Exemple (extrait du code) :

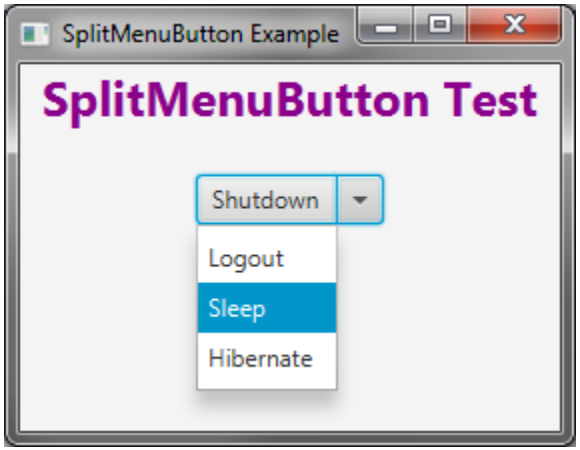
```
MenuButton mbnOptions = new MenuButton("Options");

mnuSubOpt.getItems().addAll(mniSkin, mniSize, mniColor); // Sous-menu

mbnOptions.setPopupSide(Side.LEFT);
bnOptions.getItems().addAll(mniSave, mniQuit, mnuSubOpt, mniFont);
```

SplitMenuButton

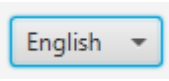


- Le composant **`SplitMenuButton`** est une sorte de combinaison du composant `Button` et du composant `MenuButton` (dont il est une sous-classe).
- La zone du bouton est divisée en 2 parties :
  - Une **zone avec un libellé** qui agit comme un bouton ordinaire en déclenchant une action lorsqu'on clique sur cette partie (code associé à la méthode `setOnAction(…)`).
  - Une **zone de menu** qui peut être activée en cliquant dessus ou en utilisant les touches "curseurs" du clavier et qui fonctionne comme le composant `MenuButton` en ouvrant un menu dans une fenêtre *popup*.
- La dernière sélection qui a été effectuée dans le menu n'est pas automatiquement associée à l'action du bouton.



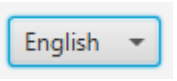


ChoiceBox [1]



- Le composant **`ChoiceBox`** permet de présenter à l'utilisateur une liste d'éléments dans laquelle il peut en sélectionner un.
- Le composant se présente au départ sous la forme d'un bouton et, lorsqu'on clique dessus, les éléments sont affichés sous la forme d'une liste déroulante.
- L'élément actuellement sélectionné est affiché en libellé lorsque la liste est *fermée (enroulée)*.
- Ce composant est à réserver pour la sélection de listes relativement courtes car il n'y a pas de *scrolling* possible.
- Par défaut, il n'y a pas d'élément sélectionné au départ.
- Le type du composant est générique `ChoiceBox<T>`, `T` étant le type des éléments de la liste.

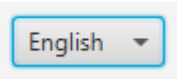
ChoiceBox [2]



- Quelques propriétés du composant `ChoiceBox` :

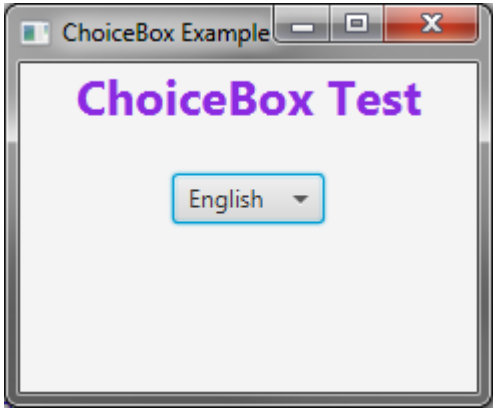
`items`	Liste des éléments à afficher. Un objet de type `ObservableList<T>` que l'on peut également passer au constructeur. La liste des éléments peut être modifiée dynamiquement (ajout, suppression, …).
`selectionModel`	Modèle de sélection des éléments de la liste. Type `SingleSelectionModel<T>`. On ne change généralement pas le modèle de sélection par défaut.
`value`	Valeur de l'élément sélectionné. Type `<T>`.
`showing`	Booléen qui indique si la liste des éléments est affichée ou non.
`converter`	Convertisseur qui indique comment l'élément est représenté. Ce convertisseur (`StringConverter<T>`) est utilisé pour convertir les éléments de type `T` en `String` (et inversement). Par défaut, c'est la méthode `toString()` de l'élément qui est utilisée.

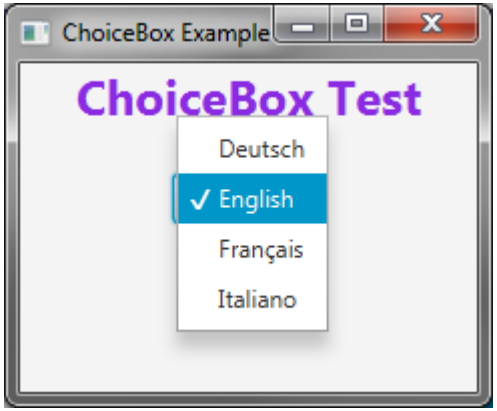
ChoiceBox [3]

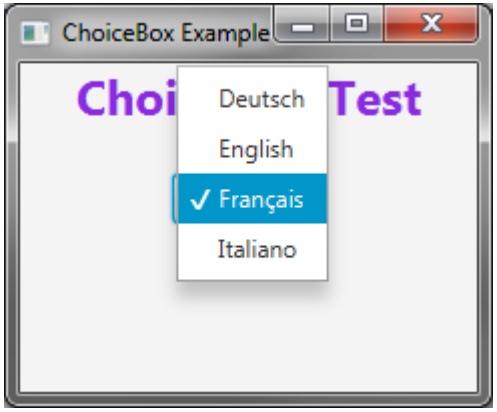


- Exemple avec une liste de chaînes de caractères (choix d'une langue).

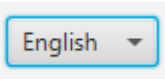
```
ChoiceBox<String> cbxLang = new ChoiceBox<String>();

cbxLang.getItems().addAll("Deutsch", "English", "Français", "Italiano");
cbxLang.getSelectionModel().select(1); // ou cbxLang.setValue("English");

root.getChildren().add(cbxLang);
```







ChoiceBox [4]



- Si l'on souhaite effectuer une action lors d'un changement de sélection, il faut enregistrer un gestionnaire d'événement sur le modèle de sélection (propriété `selectedItem` ou `selectedIndex`).

```
cbxLang.getSelectionModel().selectedItemProperty()
                           .addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> ov,
                        String s1, String s2)                        {
      System.out.println(s1+", "+s2);  // Display old and new values
      System.out.println(cbxLang.getSelectionModel().getSelectedIndex());
    }
  });
```

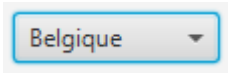
  - Identique mais avec une **expression lambda** :

```
cbxLang.getSelectionModel().selectedItemProperty()
                           .addListener((ov, s1, s2) -> {
    System.out.println(s1+", "+s2);  // Display old and new values
    System.out.println(cbxLang.getSelectionModel().getSelectedIndex());
  });
```

ComboBox [1]

- Le composant `ComboBox<T>` est assez similaire au composant `ChoiceBox` et permet de présenter à l'utilisateur une liste d'éléments (de type `T`) dans laquelle il pourra en sélectionner un.
- Une des différences réside dans le fait que `ComboBox` permet de **limiter le nombre de choix affichés** et offre automatiquement une **barre de défilement** (ascenseur) pour naviguer dans la liste.
- Une autre différence est qu'un `ComboBox` peut être **éditable** (l'utilisateur peut saisir une valeur qui ne figure pas dans la liste).
- `ComboBox` est une sous-classe de `ComboBoxBase` qui offre des propriétés et fonctionnalités de base au composant.
- La propriété `onAction` permet d'enregistrer un gestionnaire d'événement qui sera exécuté lorsque la valeur sélectionnée change.
- La fonction de saisie automatique *(autocomplétion)* n'est pas disponible par défaut et doit être codée si elle est souhaitée.

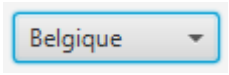
ComboBox [2]



- Quelques propriétés du composant `ComboBox` :

`items`	Liste des éléments à afficher. Un objet de type `ObservableList<T>` que l'on peut également passer au constructeur. La liste des éléments peut être modifiée dynamiquement (ajout, suppression, …).
`selectionModel`	Modèle de sélection des éléments `SingleSelectionModel<T>` (un seul élément peut être sélectionné).
`value`	Valeur de l'élément sélectionné. Type `<T>`.
`showing`	Booléen qui indique si la liste des éléments est affichée ou non.
`converter`	Convertisseur qui indique comment l'élément est représenté et, s'il est éditable, comment traduire la chaîne de caractères en objet de type `T`. Ce convertisseur (`StringConverter<T>`) est donc utilisé pour convertir les éléments de type `T` en `String` et inversement.
`editable`	Booléen qui indique si le *ComboBox* est éditable, c'est-à-dire que l'utilisateur peut saisir une valeur qui n'est pas dans la liste des éléments.

ComboBox [3]

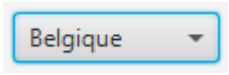


`visibleRowCount`	Nombre maximal de lignes affichées dans la liste déroulante (`Integer`). Si ce nombre est dépassé, une barre de défilement permet à l'utilisateur de parcourir la liste *(scrolling)*
`onAction`	Traitement de l'événement déclenché lors du changement de la valeur sélectionnée (`value`).

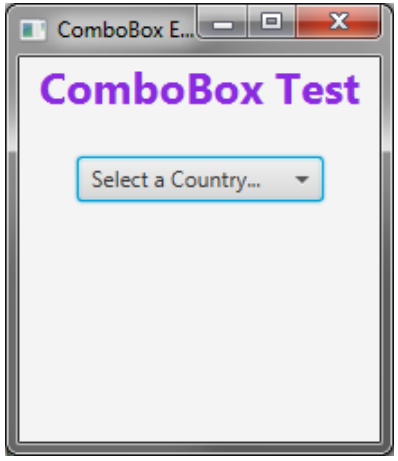
`onShowing`
`onShown`
`onHiding`
`onHidden`
	Traitement des événements associés à l'ouverture et à la fermeture du menu déroulant (juste avant et juste après l'ouverture et la fermeture de la fenêtre *popup*).

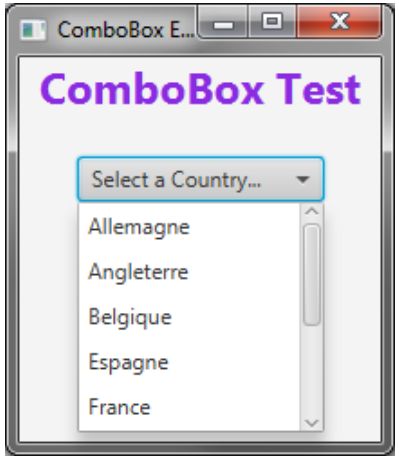
`promptText`	Texte affiché si le composant n'est pas éditable et qu'aucun élément n'est sélectionné. Par exemple à l'état initial ou après annulation de la sélection (`setValue(null);`).
`placeholder`	Composant (de type `Node`) à afficher si la liste des éléments est vide.

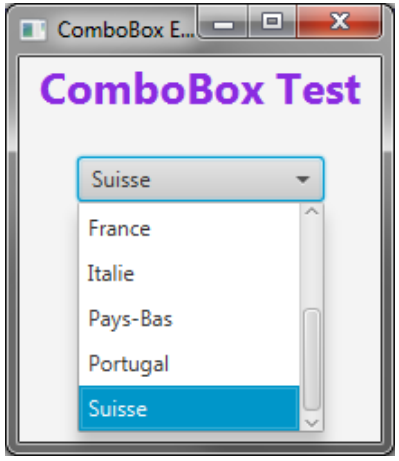
ComboBox [4]

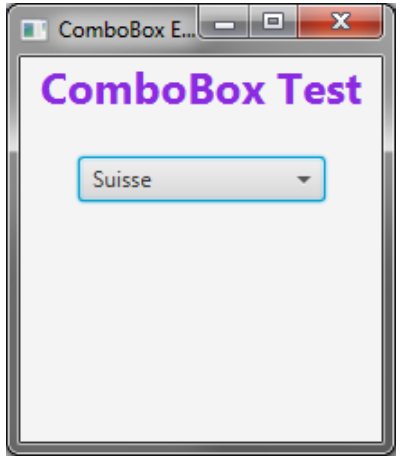


```
ComboBox<String> cbbCountry = new ComboBox<String>();

cbbCountry.getItems().addAll("Allemagne", "Angleterre", "Belgique",
                             "Espagne",   "France",     "Italie",
                             "Pays-Bas",  "Portugal",   "Suisse");

cbbCountry.setPromptText("Select a Country...");
cbbCountry.setVisibleRowCount(5);   // Max 5 éléments visibles

cbbCountry.setOnAction(event -> {
    System.out.println(cbbCountry.getValue());
  });
```









ListView [1]

- Le composant `ListView<T>` permet d'afficher un ensemble d'éléments (de type `T`) sous la forme d'une liste scrollable.
- Les éléments de la liste peuvent être affichés verticalement (par défaut) ou horizontalement.
- L'utilisateur peut sélectionner les éléments (sélection simple par défaut, mais sélection multiple possible) ou interagir avec eux.
- Comme pour les `ComboBox`, les éléments sont enregistrés en interne dans une collection de type `ObservableList` (modèle du composant) qui informe le composant `ListView` des changements intervenus.
- Les éléments de la liste peuvent potentiellement être édités par l'utilisateur (pour cela, la propriété `cellFactory` doit avoir été redéfinie).
- Le nombre d'éléments affichés dépend de la taille préférée du composant (`prefWidth`, `prefHeight`). Si nécessaire des barres de défilement *(ascenseurs)* apparaîtront pour permettre le *scrolling*.

ListView [2]

- Le peuplement de la liste peut être effectué en ajoutant les éléments avec les méthodes `add()` ou `addAll()` de la propriété `items` (comme dans les exemples pour `ChoiceBox` et `ComboBox`).
- Une autre manière de faire (courante) est d'utiliser des méthodes statiques de la classe `FXCollections` pour créer des objets de type `ObservableList<T>` qui seront assignés au composant `ListView` avec la méthode `setItems()` ou alors passés en paramètre à son constructeur.

```
ListView<String> livCountry = new ListView<String>();

ObservableList<String> countries = FXCollections.observableArrayList(
                                "Allemagne", "Angleterre", "Belgique",
                                "Espagne",   "France",     "Italie",
                                "Pays-Bas",  "Portugal",   "Suisse"   );

livCountry.setItems(countries);
```

ListView [3]

- Quelques propriétés du composant `ListView` :

`items`	Liste des éléments à afficher dans la liste. Un objet de type `ObservableList<T>` que l'on peut également passer au constructeur. La liste des éléments peut être modifiée dynamiquement (ajout, suppression, …).
`selectionModel`	Modèle de sélection des éléments `MultipleSelectionModel<T>`. - *Unique* : `setSelectionMode(SelectionMode.SINGLE)` - *Multiple* : `setSelectionMode(SelectionMode.MULTIPLE)`
`orientation`	Orientation de la liste. Type `Orientation`.
`editable`	Booléen qui indique si la *ListView* est éditable. Cela implique que les éléments (`ListCell`) doivent également être éditables ce qui nécessite de redéfinir la propriété `cellFactory`.
`placeholder`	Composant (de type `Node`) à afficher si la liste des éléments est vide.

ListView [4]

`fixedCellSize`	Valeur `Double` qui définit une hauteur (ou largeur si horizontal) fixe pour tous les éléments de la liste (calculée sinon pour chacun des éléments). Accélère l'affichage pour les longues listes.
`cellFactory`	Définition d'une fabrique d'éléments *(factory)* permettant de personnaliser totalement les éléments de la liste. Type `Callback<ListView<T>,ListCell<T>>`
`focusModel`	Modèle permettant de connaître l'élément de la liste qui possède le focus (ou son index).
`onScrollTo`	Définition de l'action à exécuter lorsque la méthode `scrollTo()` a été invoquée.

`onEditStart`
`onEditCommit`
`onEditCancel`
	Définition des actions à exécuter lors des différentes phases de l'édition des éléments (s'ils sont éditables).

ListView [5]

- Méthodes pour gérer la visibilité des éléments de `ListView` :

`scrollTo(`*index*`)`	Fait défiler la liste *(scroll)*, si nécessaire, pour que l'élément correspondant à l'indice passé en paramètre soit visible.
`scrollTo(`*object*`)`	Fait défiler la liste *(scroll)*, si nécessaire, pour que l'élément correspondant à l'objet passé en paramètre soit visible.

- Les éléments de la liste peuvent être quelconques et sont représentés par des objets de type `ListCell`.
- Il existe quelques sous-classes prédéfinies de `ListCell` qui permettent de créer facilement des listes éditables dont les éléments correspondent à des composants connus.
  - `CheckBoxListCell` : Éléments de type *CheckBox*
  - `ChoiceBoxListCell` : Elements de type *ChoiceBox*
  - `ComboBoxListCell` : Elements de type *ComboBox*
  - `TextFieldListCell` : Elements de type *TextField*

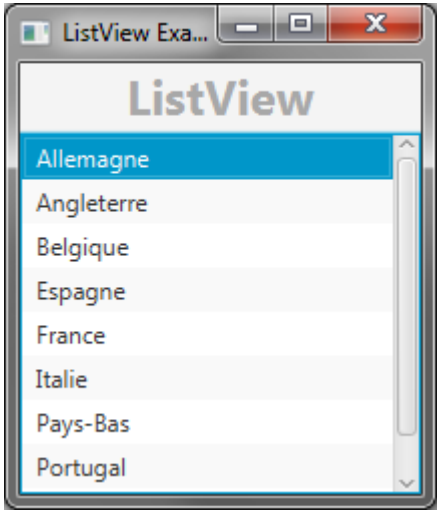
ListView [6]

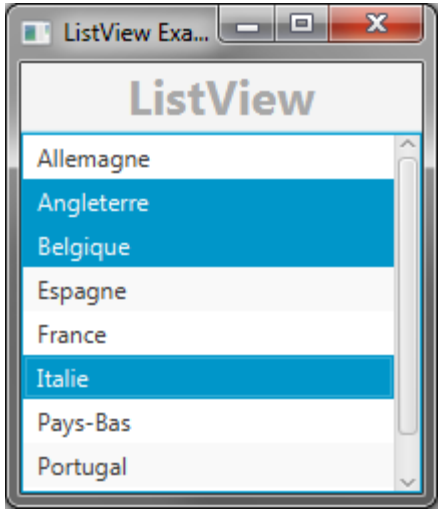
- Exemple d'utilisation d'un composant `ListView` pour permettre à l'utilisateur de sélectionner des pays (sélection multiple) et affichage sur la console de la sélection courante (à chaque changement).

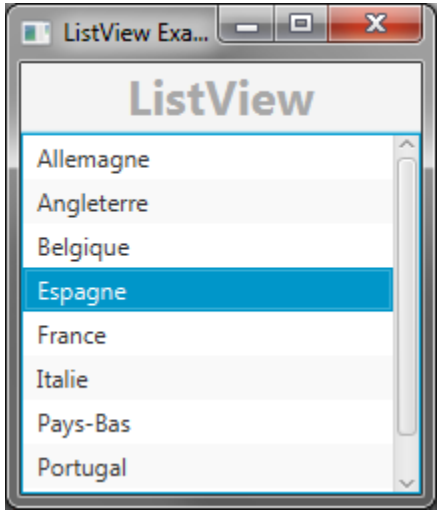
```
ListView<String> livCountry = new ListView<String>();

ObservableList<String> countries = FXCollections.observableArrayList(
                                "Allemagne", "Angleterre", "Belgique",
                                "Espagne",   "France",     "Italie",
                                "Pays-Bas",  "Portugal",   "Suisse"    );

livCountry.setItems(countries);
livCountry.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
livCountry.setFixedCellSize(22);
livCountry.setPrefSize(200, 180);

livCountry.setOnMouseClicked(event -> {
  System.out.println(livCountry.getSelectionModel().getSelectedItems());
});
```

ListView [7]

- Exécution de l'application :







- Affichage sur la console

```
[Angleterre]
[Angleterre, Belgique]
[Angleterre, Belgique, Italie]
[Espagne]
```

ListView [8]

- Exemple d'utilisation d'un composant `ListView` en plaçant une liste déroulante *(ComboBox)* comme élément éditable de la liste.

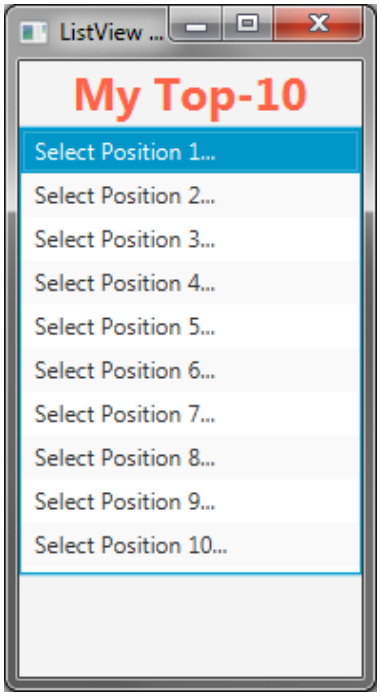
```
ObservableList<String> candidates = FXCollections.observableArrayList();
ObservableList<String> top10      = FXCollections.observableArrayList();

ListView<String> listView = new ListView<String>(top10);
listView.setPrefSize(180, 235);
listView.setEditable(true);

candidates.addAll("Mommy", "Interstallar", "Gone Girl", "Her",
                  "Boyhood", "Le sel de la terre", "Dragon 2",
                  "Night Call", "Les combattants", "The Raid 2",
                  "12 Years a Slave", "Pride", "Timbuktu", "R",
                  "Winter Sleep", "Philomena", "Nebraska"      );

for (int i = 1; i <= 10; i++) {
  top10.add("Select Position " + i + "...");  // Valeurs initiales
}

listView.setItems(top10);
listView.setCellFactory(ComboBoxListCell.forListView(candidates));
```

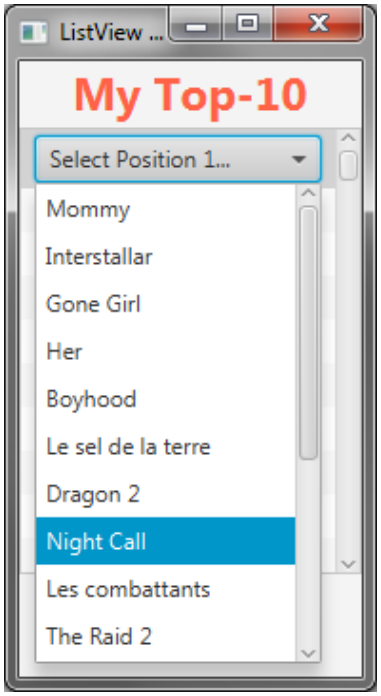
Éléments de la ComboBox

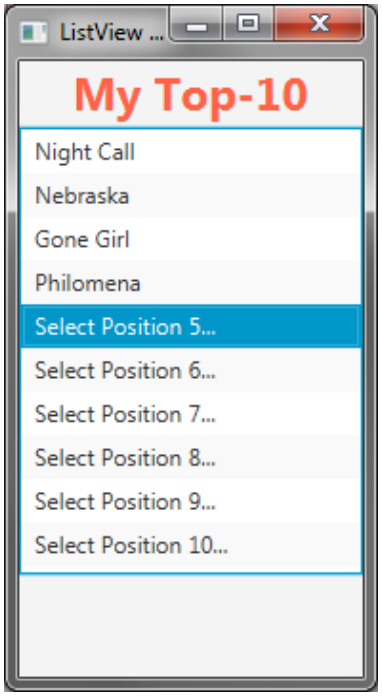
ComboBox comme ListCell

ListView [9]

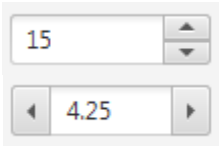
- L'utilisateur peut cliquer sur un des éléments de la liste et faire apparaître un composant qui s'apparente à un *ComboBox* qui lui permet de choisir dans la liste déroulante.
  - La valeur sélectionnée dans la liste déroulante devient la valeur de l'élément de la `ListView`.





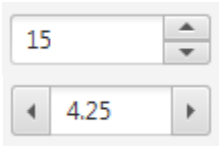


Spinner [1]



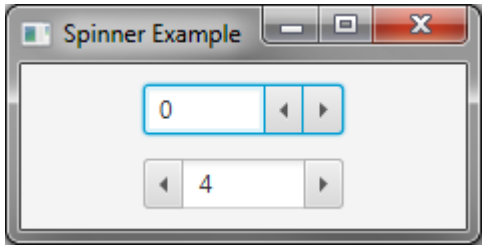
- Le composant **`Spinner<T>`** permet à l'utilisateur de sélectionner une valeur dans une liste d'éléments ordonnés (de type **`T`**).
- Seule la valeur de l'élément sélectionné est affichée dans un champ texte interne (`TextField`) nommé *éditeur* (propriété `Editor`).
- Deux boutons permettent à l'utilisateur de faire défiler les valeurs successives de la liste dans l'ordre croissant et décroissant.
- Les valeurs sélectionnables sont définies dans le modèle du composant qui est de type `SpinnerValueFactory<T>`.
  - Trois modèles sont proposés par défaut (classes internes) :
    - ⇨ `IntegerSpinnerValueFactory`
    - ⇨ `DoubleSpinnerValueFactory`
    - ⇨ `ListSpinnerValueFactory`
- Il est possible de rendre la zone de texte éditable. L'utilisateur doit alors saisir des valeurs valides (en fonction du modèle interne choisi) sinon la valeur sera adaptée [min, max] ou une exception sera générée.

Spinner [2]

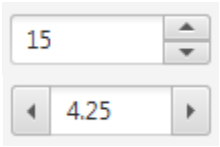


- Le défilement des valeurs peut être rendu circulaire *(wrapping).*
- L'emplacement des boutons de défilement est configurable en changeant le style CSS associé à la classe du composant.
  - En plus du style par défaut *(right vertical)*, cinq styles sont définis par des constantes de la classe `Spinner`.

```
Spinner<Integer>  sprI  = new Spinner<>(-20, 20, 0, 5);
Spinner<Double>   sprD  = new Spinner<>(1.0, 6.0, 4.0, 0.25);

sprI.getStyleClass().add(Spinner.STYLE_CLASS_SPLIT_ARROWS_VERTICAL);

sprD.getStyleClass().add(Spinner.STYLE_CLASS_SPLIT_ARROWS_HORIZONTAL);
```



Spinner [3]



- Différents constructeurs permettent de créer le composant `Spinner` en définissant simultanément le type et les valeurs enregistrées dans le modèle interne.
- Pour les modèles numériques, les valeurs minimales, maximales, initiales ainsi que les incréments *(step)* peuvent être définis.
- Exemples :

```
Spinner<Integer> sprTemp   = new Spinner<>(-20, 20, 0, 5);
Spinner<Double>  sprGrades = new Spinner<>(1.0, 6.0, 4.0, 0.25);
Spinner<String>  sprDays   = new Spinner<>(
                                FXCollections.observableArrayList(
                                        "Monday",
                                        "Tuesday",
                                        "Wednesday",
                                        "Thursday",
                                        "Friday",
                                        "Saturday",
                                        "Sunday"       ));
```

Modèle de type ListSpinnerValueFactory