Cours IHM-1
JavaFX

10 - Boîtes de dialogue (simples et spécialisées)

Définition [1]

- Les **boîtes de dialogue** sont des éléments d'une interface graphique qui se présentent généralement sous la forme d'une fenêtre affichée par une application (ou éventuellement par le système d'exploitation) dans le but :
  - d'**informer** l'utilisateur (texte, mise en garde, graphique, etc.)
  - d'**obtenir une information** de l'utilisateur (mot-de-passe, choix, etc.)
  - ou une combinaison des deux
    - ⇨ Par exemple l'informer d'un événement et lui demander de faire un choix
- Une boîte de dialogue **dépend d'une autre fenêtre** (év. d'une autre boîte de dialogue), c'est ce qui distingue ce composant d'une fenêtre indépendante (la fenêtre principale par exemple).

Modal / Non-modal

- Une boîte de dialogue peut être
  - **Modale** *(modal)* : L'utilisateur ne peut pas interagir avec la fenêtre dont la boîte de dialogue dépend avant de l'avoir fermée.
  - **Non-modale** *(modeless)* : L'utilisateur peut interagir avec la boîte de dialogue mais aussi avec la fenêtre dont la boîte de dialogue dépend (en laissant la boîte de dialogue ouverte).
- On utilisera une boîte de dialogue modale par ex. pour confirmer ou annuler une action critique (suppression de données par exemple).
  - La fenêtre principale est 'bloquée' tant que l'utilisateur n'a pas décidé.
  - La boîte de dialogue se ferme automatiquement dès la décision prise.
- Une boîte de dialogue non-modale sera utilisée par exemple pour présenter à l'utilisateur une palette d'outils qu'il pourra sélectionner et appliquer successivement sur la fenêtre principale.
  - La boîte de dialogue reste ouverte tant que l'utilisateur ne la ferme pas explicitement.

Boîtes de dialogue simples

Types de boîtes de dialogue

- Dans une application, on peut trouver
  - des boîtes de dialogue standard avec une mise en page prédéfinie et qui sont utilisées
    - ⇨ Pour informer l'utilisateur par un simple texte
    - ⇨ Pour demander quittance à l'utilisateur
    - ⇨ Pour demander à l'utilisateur de faire un choix entre plusieurs options
    - ⇨ Pour saisir une information simple (ligne de texte)
  - des boîtes de dialogue spécifiques avec
    - ⇨ Une disposition des éléments propre à l'application
    - ⇨ De nombreux composants pour représenter l'information
- La plupart des kits de développement offrent des solutions *clés en main* pour les boîtes de dialogue simples (standard).
- Les boîtes de dialogue spécifiques doivent par contre être conçues et réalisées avec les mêmes efforts que pour tout autre type d'interface (choix des conteneurs et composants, contraintes de disposition, etc.).

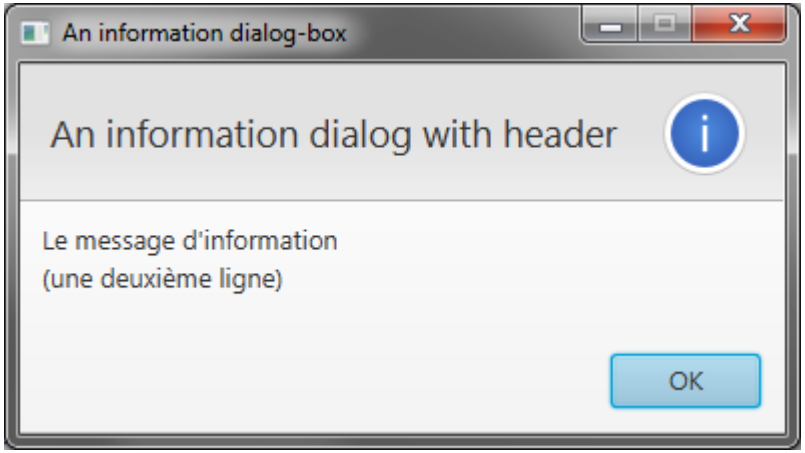
Boîtes de dialogue *JavaFX*

- Les boîtes de dialogue standard n'ont été introduites que très tardivement dans *JavaFX* (version 8u40, en 2015).
- Avant cette version, il était habituel d'utiliser les composants de type `Dialogs` de la librairie externe *ControlsFX* (qui sont désormais marqués comme étant *deprecated*).
- Les pages suivantes décrivent sommairement, et sur la base d'exemples concrets, la manière de coder les boîtes de dialogue en se basant sur les classes et composants disponibles dans *JavaFX*.

Boîte de dialogue - Alert [1]

- Boîte de dialogue `Alert` de type *"Information"* avec en-tête *(header)*.

```
Alert dialog = new Alert(AlertType.INFORMATION);

dialog.setTitle("An information dialog-box");
dialog.setHeaderText("An information dialog with header");
dialog.setContentText("Le message d'information\n" +
                      "(une deuxième ligne)");
dialog.showAndWait();
```



Boîte de dialogue - Alert [2]

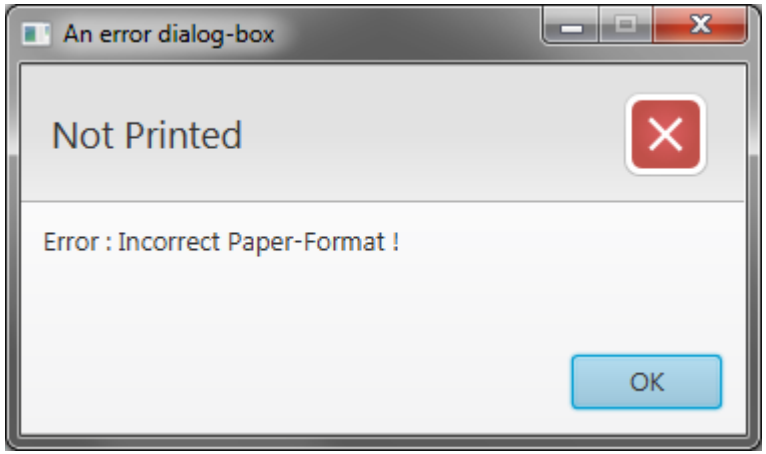
- Boîte de dialogue `Alert` de type *"Warning"* sans en-tête *(header)*.

```
Alert dialogW = new Alert(AlertType.WARNING);

dialogW.setTitle("A warning dialog-box");
dialogW.setHeaderText(null);  // No header
dialogW.setContentText("Caution : Low Level Battery !");
dialogW.showAndWait();
```

Boîte de dialogue - Alert [3]

- Boîte de dialogue `Alert` de type *"Error"*.

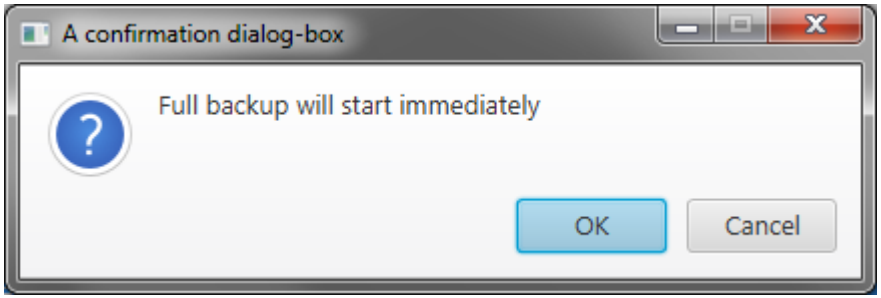
```
Alert dialogE = new Alert(AlertType.ERROR);

dialogE.setTitle("An error dialog-box");
dialogE.setHeaderText("Not Printed");
dialogE.setContentText("Error : Incorrect Paper-Format !");
dialogE.showAndWait();
```



Boîte de dialogue - Alert [4]

- Boîte de dialogue `Alert` de type *"Confirmation"*.

```
Alert dialogC = new Alert(AlertType.CONFIRMATION);

dialogC.setTitle("A confirmation dialog-box");
dialogC.setHeaderText(null);
dialogC.setContentText("Full backup will start immediately");
Optional<ButtonType> answer = dialogC.showAndWait();

if (answer.get() == ButtonType.OK) {
  System.out.println("User chose OK");
}
else {
  System.out.println("User chose Cancel or closed the dialog-box");
}
```



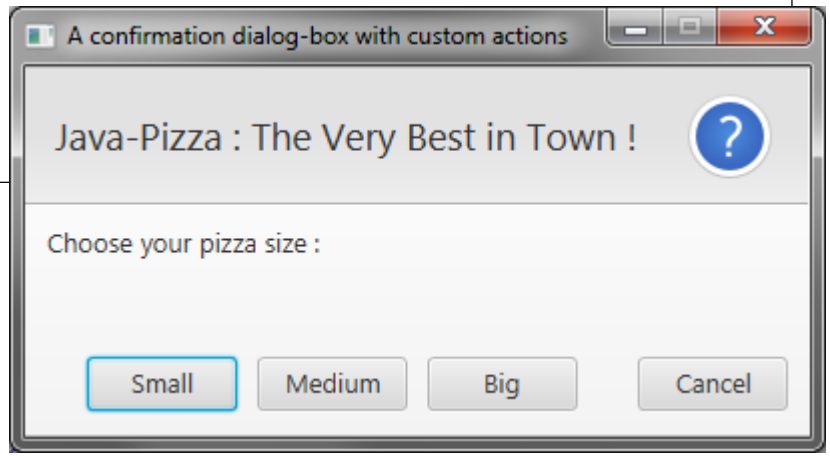
Boîte de dialogue - Alert [5]

- Boîte de dialogue `Alert` de type *"Confirmation"* avec options personnalisées.

```
Alert dBox = new Alert(AlertType.CONFIRMATION);

dBox.setTitle("A confirmation dialog-box with custom actions");
dBox.setHeaderText("Java-Pizza : The Very Best in Town !");
dBox.setContentText("Choose your pizza size :");

ButtonType btnSmall  = new ButtonType("Small");
ButtonType btnMedium = new ButtonType("Medium");
ButtonType btnBig    = new ButtonType("Big");
ButtonType btnCancel = new ButtonType("Cancel", ButtonData.CANCEL_CLOSE);

dBox.getButtonTypes().setAll(btnSmall, btnMedium, btnBig, btnCancel);
```

Boîte de dialogue - Alert [6]

```
Optional<ButtonType> choice = dBox.showAndWait();

if (choice.get() == btnSmall) {
  System.out.println("User chose Small");
}
else if (choice.get() == btnMedium) {
  System.out.println("User chose Medium");
}
else if (choice.get() == btnBig) {
  System.out.println("User chose Big");
}
else {
  System.out.println("Cancel or Close");
}
```



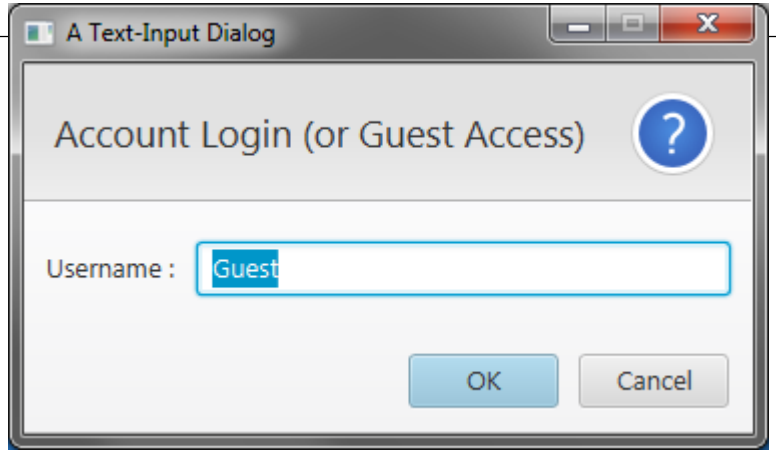
Saisie de texte - TextInputDialog [1]

- Boîte de *dialogue* `TextInputDialog` pour saisir une ligne de texte.

```
TextInputDialog inDialog = new TextInputDialog("Guest");

inDialog.setTitle("A Text-Input Dialog");
inDialog.setHeaderText("Account Login (or Guest Access)");
inDialog.setContentText("Username :");

Optional<String> textIn = inDialog.showAndWait();

//--- Get response value (traditional way)
if (textIn.isPresent()) {
  System.out.println("Login name = " + textIn.get());
}
```

Valeur par défaut

Retourne false *si l'utilisateur presse* Cancel *ou ferme la boîte de dialogue*



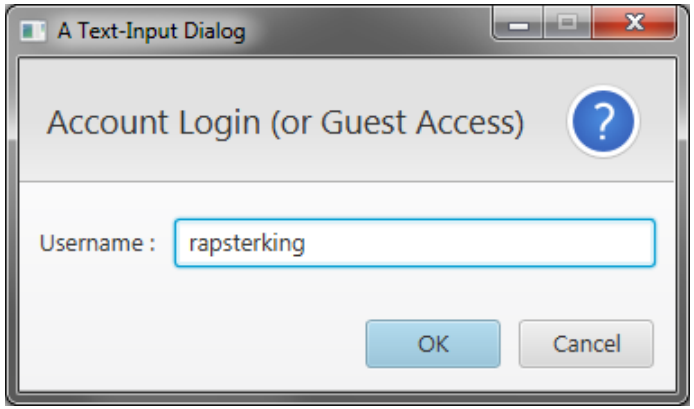
Saisie de texte - TextInputDialog [2]

- Le texte saisi par l'utilisateur peut aussi être récupéré en utilisant la méthode `ifPresent()` qui prend en paramètre un objet de type `Consumer<T>` que l'on peut instancier avec une expression lambda.

  L'interface fonctionnelle `Consumer<T>` possède la méthode abstraite `void accept(T t)` qui effectue une opération avec la valeur `t` passée en paramètre (pas de valeur de retour, opération par effet de bord).

```
Optional<String> textIn = inDialog.showAndWait();

//--- Get response value (with lambda expression)
textIn.ifPresent(txt -> System.out.println("Login name = " + txt));
```

Expression lambda non exécutée si l'utilisateur presse Cancel *ou ferme la boîte de dialogue*



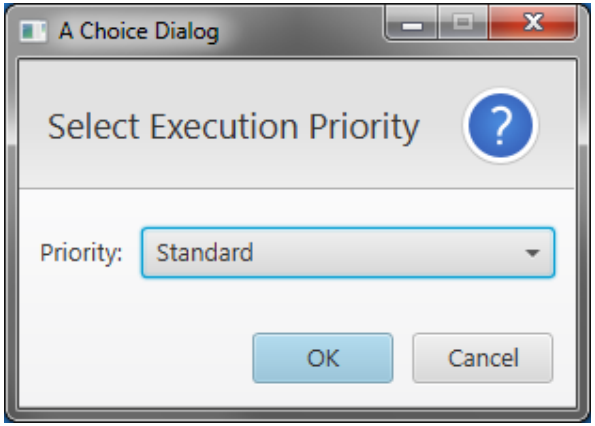
Sélection - ChoiceDialog [1]

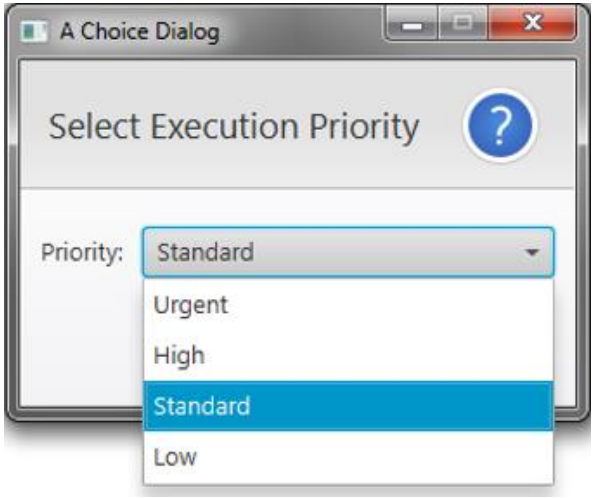
- Boîte de *dialogue* `ChoiceDialog` pour saisir un choix (dans une liste).

```
String[]  choices = {"Urgent", "High", "Standard", "Low"};

ChoiceDialog<String> cDial = new ChoiceDialog<>(choices[2], choices);

cDial.setTitle("A Choice Dialog");
cDial.setHeaderText("Select Execution Priority");
cDial.setContentText("Priority:");

Optional<String> selection = cDial.showAndWait();

selection.ifPresent(str -> System.out.println("Selection:" + str));
```

Valeur par défaut

Expression lambda non exécutée si l'utilisateur presse Cancel *ou ferme la boîte de dialogue*



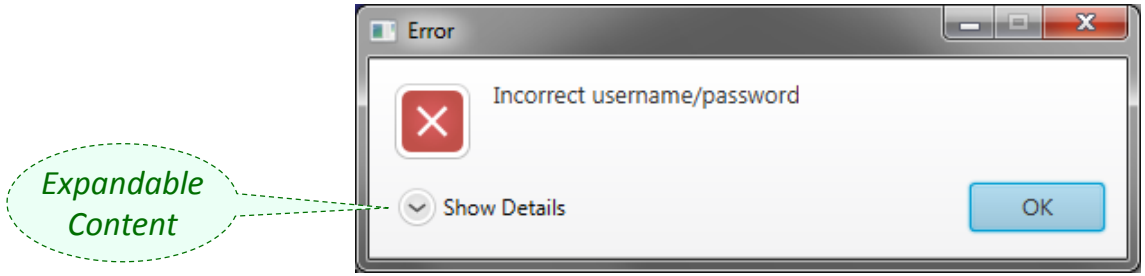


Boîte de dialogue personnalisée [1]

- Il est possible de personnaliser des boîtes de dialogue *(custom dialog)* en utilisant directement la classe `Dialog` qui est la classe parente de `Alert`, `TextInputDialog` et `ChoiceDialog`.
- En plus du texte principal défini par la propriété `contentText` (de type `String`), il est possible, pour toutes les boîtes de dialogue, de définir un contenu étendu (affiché par *"Show Details"*) représenté par la propriété `expandableContent` (de type `Node`) de l'objet interne `DialogPane`.
- On peut définir ce contenu étendu ainsi :

```
Alert mBox = new Alert(AlertType.ERROR);
mBox.getDialogPane().setExpandableContent(node);
```

Peut correspondre à tout un graphe de scène



Boîte de dialogue personnalisée [2]

- Par défaut, toutes les boîtes de dialogues standards (qui héritent de `Dialog`) sont **modales**.
- Pour créer une boîte de dialogue non-modale, on peut invoquer la méthode `initModality()` :

```
//--- To create a modeless dialog box
Alert mBox = new Alert(AlertType.ERROR);
mBox.initModality(Modality.NONE);
```

- L'icône par défaut (placée dans la barre de titre de la fenêtre *popup*) peut être remplacée par une icône personnalisée :

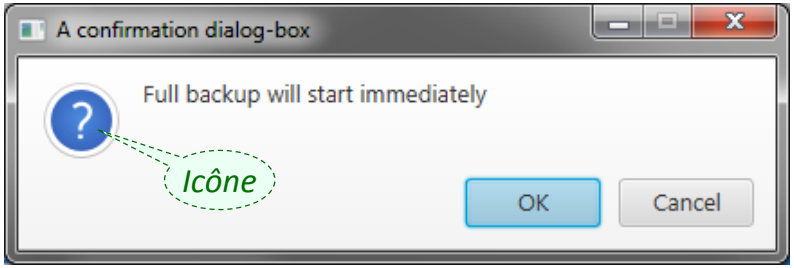
```
//--- To add a custom title-bar icon
Stage dStage = (Stage)(cDial.getDialogPane().getScene().getWindow());
dStage.getIcons().add(new Image("/resources/warn_1.png"));
```

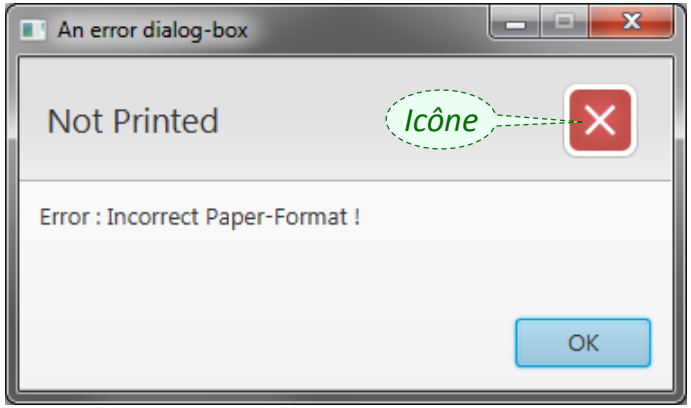
Boîte de dialogue personnalisée [3]

- Pour changer l'icône principale qui est placée à gauche du contenu de la boîte de dialogue ou dans son en-tête *(header)*, on peut utiliser la méthode `setGraphic()`.

```
//--- Add a custom content icon
cDial.setGraphic(new ImageView("/resources/warn_1.png"));
```

- Par défaut, cette icône est déterminée par le type de boîte de dialogue ou par le type d'alerte (`AlertType` : `ERROR`, `WARNING`, …).





Boîtes de dialogue spécialisées

FileChooser [1]

- L'utilitaire **FileChooser** permet d'ouvrir une boîte de dialogue permettant à l'utilisateur de naviguer dans l'arborescence des fichiers de la machine cible et de **sélectionner un ou plusieurs fichiers** (pour sélectionner un répertoire, il faut utiliser `DirectoryChooser`).
- Il s'agit d'une boîte de dialogue spécifique au système d'exploitation de la machine cible. Le *look & feel* est indépendant de *JavaFX* (il correspond en principe au *look & feel* natif).
- Ce composant peut être utilisé pour :
  - Naviguer et **sélectionner un seul fichier** ⇨ `showOpenDialog()`
  - Naviguer et **sélectionner plusieurs fichiers** ⇨ `showOpenMultipleDialog()`
  - Naviguer et **sauvegarder un fichier** ⇨ `showSaveDialog()`
- L'invocation de `FileChooser` est généralement liée à l'action d'un bouton, d'une option de menu ou autre action de l'utilisateur.

FileChooser [2]

- Exemple d'utilisation de `FileChooser` avec :
  - Définition d'un **titre** pour la fenêtre *popup*
  - Définition d'un **répertoire initial** (point de départ de la navigation) basé sur des propriétés du système
  - Définition de **filtres** basés sur les extensions des fichiers

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("FileChooser Example");

File homeDir = new File(System.getProperty("user.home"));
fileChooser.setInitialDirectory(homeDir);

fileChooser.getExtensionFilters().addAll(
      new ExtensionFilter("Text Files" , "*.txt"),
      new ExtensionFilter("Image Files", "*.png", "*.jpg", "*.gif"),
      new ExtensionFilter("Audio Files", "*.wav", "*.mp3", "*.aac"),
      new ExtensionFilter("All Files"  , "*.*"));
```

FileChooser [3]

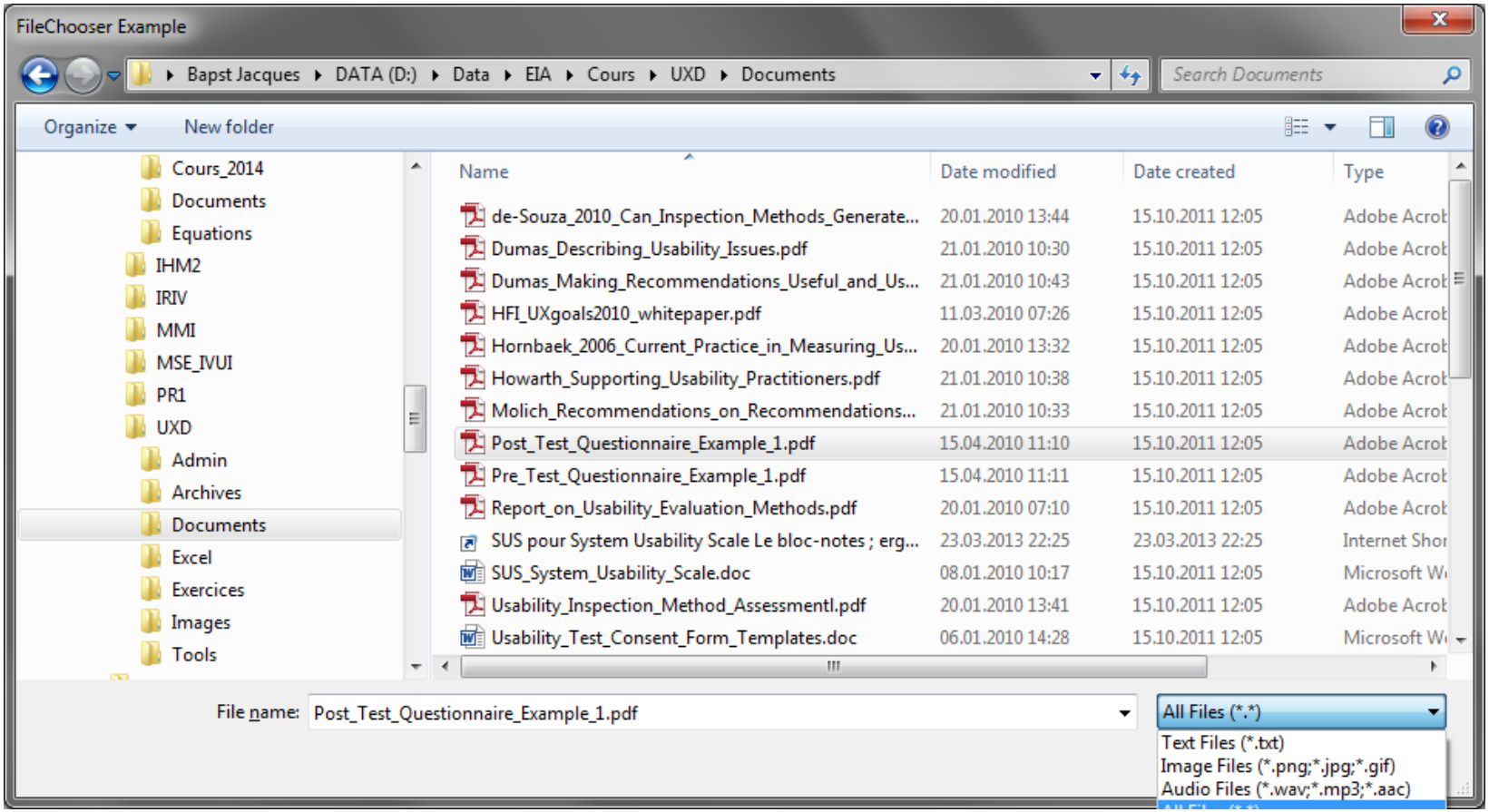
- La fenêtre *popup* s'ouvre lors de l'invocation de la méthode `show…()` qui retourne le fichier sélectionné ou une référence `null` si l'utilisateur a pressé sur *Cancel* ou a fermé la fenêtre.

```
File selectedFile = fileChooser.showOpenDialog(primaryStage);

if (selectedFile != null) {
  try {
    //--- Open the file with associated application
    Desktop.getDesktop().open(selectedFile);
  }
  catch (Exception e) {
    System.err.println("ERROR: Unable to open the file");
  }
}
```

Tente d'ouvrir le fichier avec l'application associée

FileChooser [4]

- Affichage (sous *Windows*) :



FileChooser [5]

- Pour permettre une sélection multiple, il faut utiliser la méthode `showOpenMultipleDialog()` qui retourne une liste de fichiers ou une référence `null` si l'utilisateur a pressé sur *Cancel* ou a fermé la fenêtre.

```
List<File> selectedFiles = fileChooser.showOpenMultipleDialog(stage);

if (selectedFiles != null) {
  for (File f : selectedFiles) {
    System.out.println(f);
  }

  selectedFiles.forEach(f -> System.out.println(f));

  selectedFiles.forEach(System.out::println);
}
```

Codage classique

Avec expression lambda

Avec référence de méthode

DirectoryChooser [1]

- L'utilitaire **DirectoryChooser** permet d'ouvrir une boîte de dialogue permettant à l'utilisateur de naviguer dans l'arborescence des fichiers de la machine cible et de **sélectionner un répertoire**.
- Il s'agit d'une boîte de dialogue spécifique au système d'exploitation de la machine cible. Le *look & feel* est indépendant de *JavaFX* (il correspond au *look & feel* natif).
- Ce composant ne comporte qu'une seule méthode de sélection `showDialog()`, qui retourne un objet de type `File` représentant un répertoire.

DirectoryChooser [2]

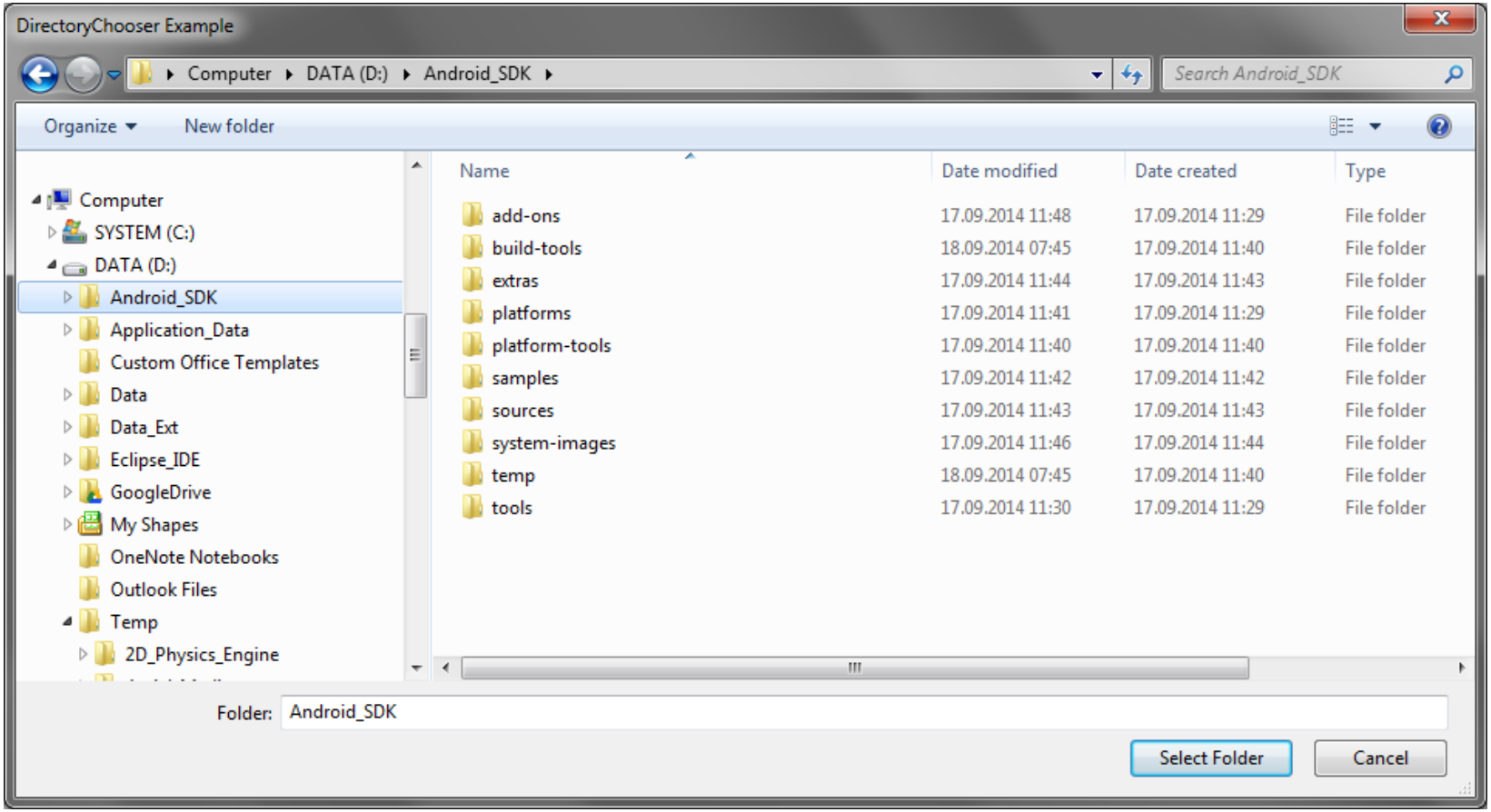
- Exemple :

```
DirectoryChooser dirChooser = new DirectoryChooser();
dirChooser.setTitle("DirectoryChooser Example");
dirChooser.setInitialDirectory(new File("D://Temp"));

File selectedFile = dirChooser.showDialog(primaryStage);

if (selectedFile != null) {
  try {
    Desktop.getDesktop().open(selectedFile);
  }
  catch (Exception e) {
    e.printStackTrace();
  }
}
```

Tente d'ouvrir le gestionnaire de fichiers de l'OS (file manager)

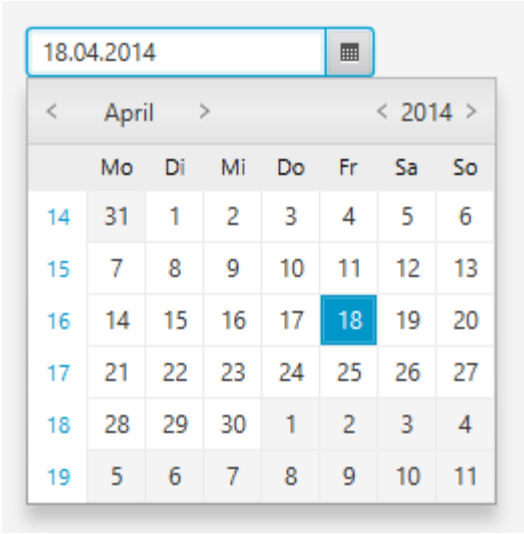
DirectoryChooser [3]

- Affichage (sous *Windows*) :



DatePicker [1]

- La classe **DatePicker** représente un composant qui permet à l'utilisateur de **sélectionner une date** dans un calendrier qui est affiché en fenêtre *popup*.
- Le composant représente une forme de *ComboBox* éditable (il a comme classe parente `ComboBoxBase`) et la date sélectionnée par l'utilisateur peut être consultée avec la méthode `getValue()` qui retourne un objet de type `LocalDate`.



DatePicker [2]

- Exemple d'utilisation du composant :

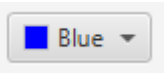
```
. . .
DatePicker dPicker = new DatePicker();

root.getChildren().add(dPicker);

dPicker.setOnAction(e -> {
            LocalDate date = dPicker.getValue();
            System.out.println("Selected date: " + date);
            . . .
            int day   = date.getDayOfMonth();   // 1..31
            int month = date.getMonthValue();   // 1..12
            int year  = date.getYear();
            . . .
          });
. . .
```

Ajout du composant dans le graphe de scène

ColorPicker [1]

- La classe **ColorPicker** représente un composant qui permet à l'utilisateur de **sélectionner une couleur** dans une palette ou selon différents espaces de couleur (*HSB*, *RBG*, *Web*, …).
- Le composant représente une forme de *ComboBox* éditable (il a comme classe parente `ComboBoxBase`) et la couleur sélectionnée par l'utilisateur peut être consultée avec la méthode `getValue()`.
- L'utilisateur peut définir et enregistrer des couleurs personnalisées *(custom colors)* qui peuvent être récupérées avec la méthode `getCustomColors()` qui retourne une liste de couleurs.
- Une couleur par défaut peut être passée en paramètre au constructeur de la classe (par défaut `Color.WHITE`).

ColorPicker [2]



ColorPicker [3]

- Exemple d'utilisation du composant :

```
. . .
ColorPicker cPicker = new ColorPicker(Color.BLUE);

root.getChildren().add(cPicker);

cPicker.setOnAction(e -> {
        Color c = cPicker.getValue();
        . . .
        //--- Display hexadecimal values (#rrggbbaa)
        System.out.println("Selected color: " + c);

        // Display decimal values RGB (0.0...1.0)
        System.out.println("Selected color = " + c.getRed()   + ", "
                                               + c.getGreen() + ", "
                                               + c.getBlue()        );
        . . .
    });
. . .
```

Ajout du composant dans le graphe de scène