Université de Limoges

R2.02

JavaFX™

Notions avancées

Objectifs du cours

- Rappels…
- Divers composants JavaFX
- Les propriétés et le mécanisme de binding

Rappels

- La programmation évènementielle
- Le design pattern Observer
- Les évènements dans JavaFX
  - Les types d'évènements
  - La propagation de ces évènements
- Les écouteurs
  - Les Event Filters (les différentes manières de les instancier)
  - Les Event Handlers
  - Les "convenience methods"

Divers composants JavaFX

- Les images et les dessins
- Les menus
- Les boites de dialogues
- Les propriétés et le mécanisme de binding

Charger une image

- On peut **charger une image** (locale ou distante)
- Pour cela on s'appuie sur la classe javafx.scene.image.**Image**

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html

- Exemples :
  - Chargement à partir d'une ressource présente dans le classpath

```
Image img1 = new Image("file:/C:/...chemin/uneImage.png");
// ou
Image img2 = new Image(getClass().getResourceAsStream("i.png"));
```

  - Chargement à partir d'une ressource présente sur le disque local

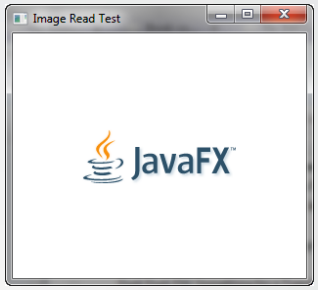
```
File file = new File("C:\\Users\\Paul\\Images\\myphoto.jpg");
Image img1 = new Image(new FileInputStream(file));
// ou
String localUrl = file.toURI().toURL().toString();
Image img2 = new Image(localUrl);
```

  - Chargement à partir d'une ressource distante (sur un serveur web)

```
String remoteUrl = "http://monserveur.com/maphoto.jpg";
Image remoteImage = new Image(remoteUrl, true); // charge en tâche de fond
```

Il peut y avoir un second paramètre de type boolean permettant d'indiquer si le chargement doit se faire en tâche de fond (true) ou pas (false)

Afficher une image



- On s'appuie sur le nœud **ImageView**

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html

- C'est un wrapper d'objet image, il faut juste la lui fournir

```
ImageView imageView = new ImageView(img);
```

- Ensuite il faut rattacher ce nœud au graphe de scène
- Comme c'est un nœud JavaFX, on peut lui appliquer divers effets (changer la taille, rotation, réduire la zone)

- On peut modifier l'icône de notre application :

```
primaryStage.getIcons().add(image);
```

- Ou l'icône d'un composant (un Button par exemple) :

```
btn.setGraphic(new ImageView(img));
```

Dessiner des formes primitives

- JavaFX permet de dessiner diverses formes 2D (shapes) :

  Line, Rectangle, Circle, Ellipse, Polygon, Polyline, CubicCurve, QuadCurve, Arc, Path, Text, SVGPath

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape.html

- Pour créer une de ces formes, il suffit :
  - d'instancier la classe correspondante à la forme
  - de positionner ses propriétés (dépend de la forme)
  - de l'ajouter à un containeur (on utilise généralement un **Group**) et le rattacher au graphe de scène

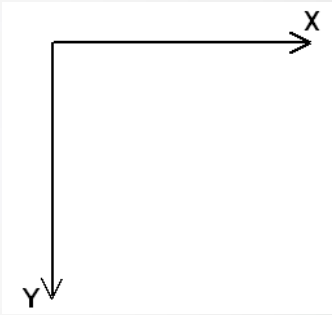
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Group.html

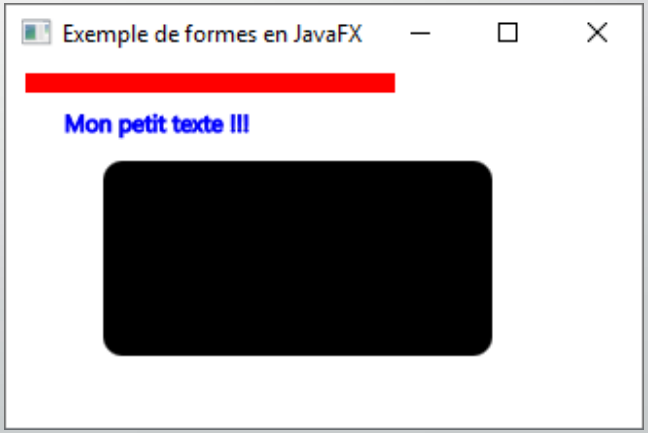
Propriétés des Shapes

- Sur chaque forme, on dispose de différentes propriétés :
  - position (coordonnées en x et y)
  - taille ou dimension
  - couleur (pleine ou bien avec gradient)
  - autres selon la forme…
- On peut également appliquer :
  - des opérations entre plusieurs formes (union, intersection, etc)
  - des effets visuels (flou, bruit gaussien, ombre interne ou externe, etc)
  - des animations

Shape : exemple

```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.WHITE);

// on créé une ligne rouge
Line ligneRouge = new Line(10, 10, 200, 10);
// on positionne les propriétés
ligneRouge.setStroke(Color.RED);
ligneRouge.setStrokeWidth(10);

root.getChildren().add(ligneRouge);

// on créé un texte bleu
Text petitText = new Text("Mon petit texte !!!");

petitText.setX(30);
petitText.setY(35);
petitText.setStroke(Color.BLUE);

root.getChildren().add(petitText);

// on créé un rectangle noir
Rectangle rect = new Rectangle();

rect.setX(50);
rect.setY(50);
rect.setWidth(200);
rect.setHeight(100);
rect.setArcWidth(20);
rect.setArcHeight(20);

root.getChildren().add(rect);

primaryStage.setTitle("Exemple de formes en JavaFX");
primaryStage.setScene(scene);
primaryStage.show();
```





Autre façon de dessiner : le Canvas

- Le nœud de type **Canvas** est un support pour dessiner
- On peut appliquer un certain nombre de commandes graphiques fournies par le **GraphicsContext** (c'est le contexte graphique de notre dessin)
- Chaque commande pousse les paramètres nécessaires dans un **buffer** qui appliquera le rendu désiré sur le dessin
- Chaque Canvas contient un unique GraphicsContext et un unique buffer
- Le **GraphicsContext** garde une pile des états des objets du contexte (qu'il peut sauver et restaurer à n'importe quel moment)

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html

https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

Similaire à l'élément html 5 <canvas> utilisé pour dessiner des graphiques sur une page web

API de Canvas

- Pour dessiner, il suffit de :
  - créer un nœud de type Canvas

```
Canvas canvas = new Canvas(300, 250);
```

  - l'ajouter à un containeur (on utilise généralement un **Group**)
  - récupérer le contexte graphique

```
GraphicsContext gc = canvas.getGraphicsContext2D();
```

  - utiliser ce contexte pour appliquer des commandes graphiques
    - ***stoke*** : pour dessiner des traits
    - ***fill*** : pour remplir des formes

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html

API de Canvas

Quelques exemples de commandes graphiques :

- dessiner une ligne : **strokeLine**(x1, y1, x2, y2)
- dessiner un rectangle : **strokeRect**(x, y, width, height)
- remplir une forme ovale : **fillOval**(x, y, width, height)
- dessiner une image : **drawImage**(image, x, y, width, height)
- dessiner du texte : **strokeText**(x1, y1, x2, y2)

  attention : il ne s'agit pas d'un Label mais bien d'un dessin !

  on peut changer la police de caractères avec **setFont**(font)

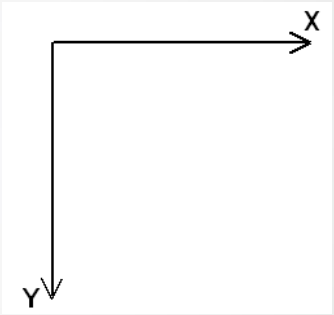
Une couleur est définie pour les opérations de type **stroke** et une autre couleur est définie pour les opérations de type **fill** :

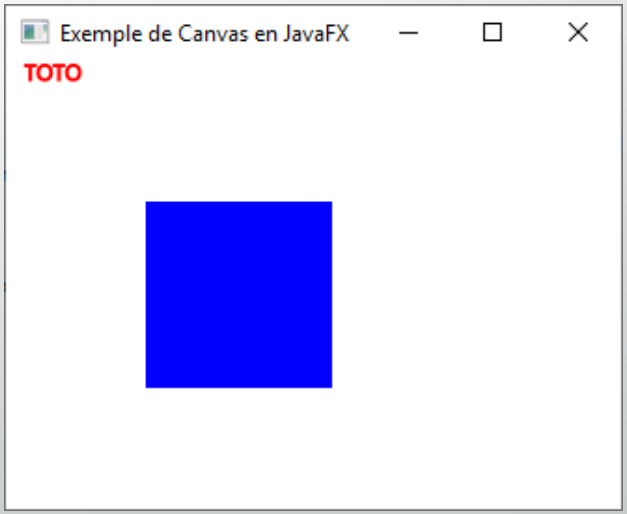
setStroke(couleur**)** : change la couleur pour les opérations de type **stroke**

setFill(couleur**)** : change la couleur pour les opérations de type **fill**

Canvas : exemple

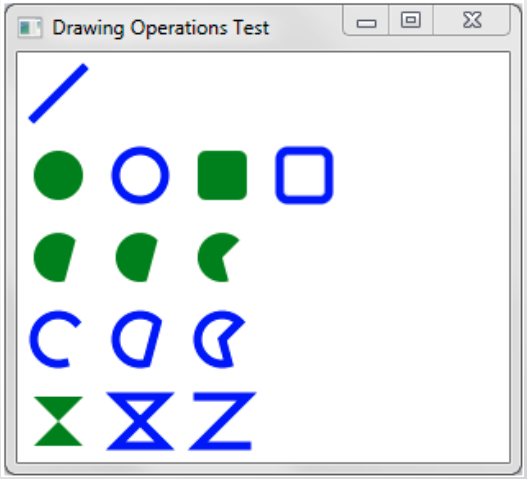
```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.WHITE);

Canvas canvas = new Canvas(300, 250);

GraphicsContext gc = canvas.getGraphicsContext2D();
gc.setStroke(Color.RED);
gc.strokeText("TOTO", 10, 10);

gc.setFill(Color.BLUE);
gc.fillRect(75,75,100,100);

root.getChildren().add(canvas);

primaryStage.setTitle("Exemple de Canvas en JavaFX");
primaryStage.setScene(scene);
primaryStage.show();
```





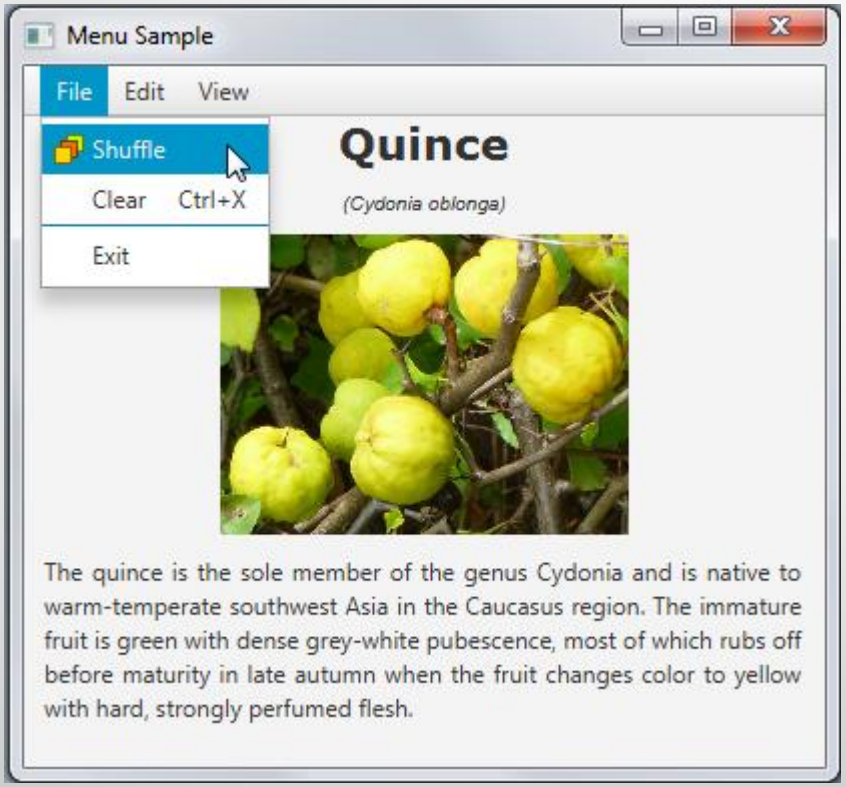
Canvas : exemple bis

```
gc.setFill(Color.GREEN);
gc.setStroke(Color.BLUE);
gc.setLineWidth(5);
gc.strokeLine(40, 10, 10, 40);
gc.fillOval(10, 60, 30, 30);
gc.strokeOval(60, 60, 30, 30);
gc.fillRoundRect(110, 60, 30, 30, 10, 10);
gc.strokeRoundRect(160, 60, 30, 30, 10, 10);
gc.fillArc(10, 110, 30, 30, 45, 240, ArcType.OPEN);
gc.fillArc(60, 110, 30, 30, 45, 240, ArcType.CHORD);
gc.fillArc(110, 110, 30, 30, 45, 240, ArcType.ROUND);
gc.strokeArc(10, 160, 30, 30, 45, 240, ArcType.OPEN);
gc.strokeArc(60, 160, 30, 30, 45, 240, ArcType.CHORD);
gc.strokeArc(110, 160, 30, 30, 45, 240, ArcType.ROUND);
gc.fillPolygon(new double[]{10, 40, 10, 40},
               new double[]{210, 210, 240, 240}, 4);
gc.strokePolygon(new double[]{60, 90, 60, 90},
                 new double[]{210, 210, 240, 240}, 4);
gc.strokePolyline(new double[]{110, 140, 110, 140},
                  new double[]{210, 210, 240, 240}, 4);
```



https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

Les menus


Menu Sample
File
Edit
View
Shuffle
Clear Ctrl+X
Exit
Quince
(Cydonia oblonga)
The quince is the sole member of the genus Cydonia and is native to warm-temperate southwest Asia in the Caucasus region. The immature fruit is green with dense grey-white pubescence, most of which rubs off before maturity in late autumn when the fruit changes color to yellow with hard, strongly perfumed flesh.

Les menus

On peut créer une barre de menu à l'aide des classes suivantes :

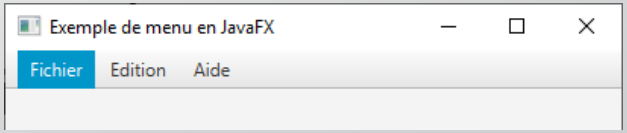
- **MenuBar** ou **ContextMenu** (menu contextuel)
  - **Menu**
    - **MenuItem**
    - **CheckMenuItem**
    - **RadioMenuItem**
    - **CustomMenuItem**
      - **SeparatorMenuItem**

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/menu_controls.htm

Créer une barre de menu

- A la racine, vous aurez :
  - soit un **MenuBar** (une barre de menu)
  - soit un **ContextMenu** (un menu contextuel)
- Ce composant contient l'ensemble du menu, et devra être placé dans un containeur (un Layout Pane)
- Il contient des objets **Menu** (c'est-à-dire les gros titres)

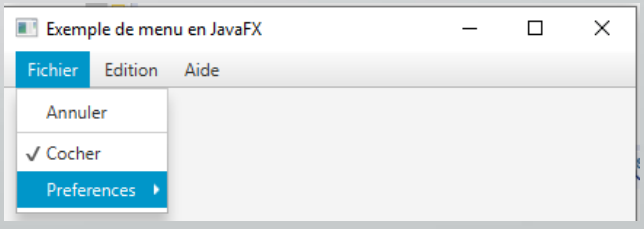
```
// création de la barre de menu
MenuBar barreDeMenu = new MenuBar();

// création de 3 Menu : Fichier, Edition, Aide
Menu menuFichier = new Menu("Fichier");
Menu menuEdition = new Menu("Edition");
Menu menuAide    = new Menu("Aide");


// Ajout des 3 menus (Fichier, Edition, Aide) dans la barre de menus
barreDeMenu.getMenus().addAll(menuFichier, menuEdition, menuAide);

// creation d'un Layout Pane de type VBox
VBox root = new VBox();
root.getChildren().addAll(barreDeMenu);
```



Insérer des éléments

- Chaque objet **Menu** est lui-même un containeur d'items, il peut contenir différents éléments :
  - un menu imbriqué (ou sous-menu) : **Menu**
  - une option dans le menu : **MenuItem**
  - une option spécialisée :
    - **CheckMenuItem** : option à cocher
    - **RadioMenuItem** : choix entre plusieurs options (d'un **ToggleGroup**)
    - **CustomMenuItem** : composant personnalisé, il peut encapsuler n'importe quel nœud et le proposer en tant qu'option de menu
    - **SeparatorMenuItem** (hérite de CustomMenuItem) : séparateur

```
// création du Menu Fichier
Menu menuFichier = new Menu("Fichier");

// création d'items de menu
MenuItem miAnnuler = new MenuItem("Annuler");
SeparatorMenuItem spi1 = new SeparatorMenuItem();
CheckMenuItem cmiCocher = new CheckMenuItem("Cocher");
Menu menuPreference = new Menu("Preferences");

// on rattache tous ces items au menu Fichier
menuFichier.getItems().addAll(miAnnuler, spi1, cmiCocher, menuPreference);
```



Manipuler les éléments

- Chaque **MenuItem** peut être :
  - actionné (ou sélectionné selon la nature du composant)
  - dé-sélectionné : `.setSelected(false)`
  - activé ou dé-sactivé : `.setDisable(true)`
- On peut aussi lui associer une icône :

```
.setGraphic( new ImageView(new Image("icone.png")) );
```

- Différentes manières pour actionner un élément de menu :
  - clic souris : on peut attacher un écouteur

```
.setOnAction(new EventHandler<ActionEvent>(){...});
```

  - raccourci clavier (avec Accelerator et KeyCombination) :

```
.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
```

Les boites de dialogues

Les boites de dialogues

- Elles sont présentes seulement depuis 2015 (arrivées avec la version JavaFX qui accompagne la JDK 8u40)
- Avant on était obligé de passer par **Control FX** (une librairie complémentaire de composants UI Control)

  http://fxexperience.com/controlsfx/features/
- A présent on peut trouver en natif :
  - **des boites de dialogues simples :** message d'information, d'avertissement, de confirmation, d'erreur, etc
  - **des boites de dialogues spécialisées :** message d'invitation de saisie, choix de fichiers, choix de date, choix de couleurs
- Elles héritent de la classe **Dialog**, et sont rattachées à un containeur de type **DialogPane**

  https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Dialog.html

Les boites de dialogues simples

Elles se basent sur l'objet **Alert**, elles possèdent :

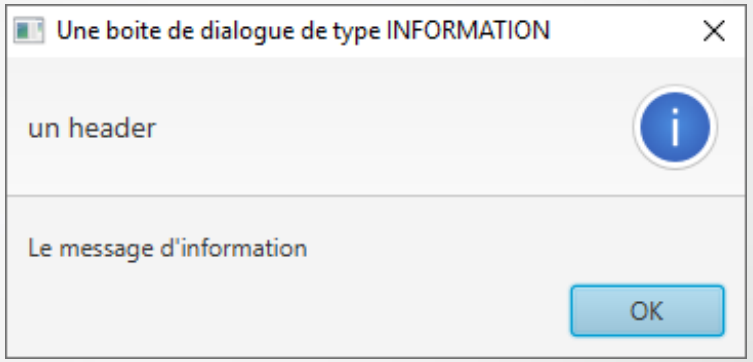
- un type : **AlertType**

  (CONFIRMATION, ERROR, INFORMATION, NONE, WARNING)
- un titre : **title**
- un texte dans l'entête (peut être vide) : **headerText**
- un texte dans le contenu du corps : **contentText**
- éventuellement des boutons : **buttonTypes**

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Alert.html

Exemple Alert 1/3

- **Alert** de type "*INFORMATION*" avec un en-tête *(header)*




```
Alert dialog = new Alert(AlertType.INFORMATION);
dialog.setTitle("Une boite de dialogue de type INFORMATION");
dialog.setHeaderText("un header");
dialog.setContentText("Le message d'information");


dialog.showAndWait();
```

Exemple Alert 2/3

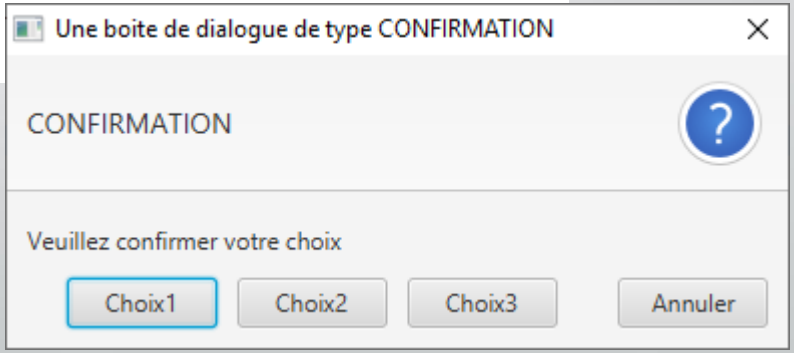
- **Alert** de type *"WARNING" sans header*


```
Alert dialog = new Alert(AlertType.WARNING);
dialog.setTitle("Une boite de dialogue de type WARNING");
dialog.setHeaderText(null); // pas de header
dialog.setContentText("Le message d'avertissement");

dialog.showAndWait();
```

Exemple Alert 3/3

- **Alert** de type *"CONFIRMATION" avec options personnalisées*

```
Alert dialog = new Alert(AlertType.CONFIRMATION);
dialog.setTitle("Une boite de dialogue de type CONFIRMATION");
dialog.setHeaderText("CONFIRMATION");
dialog.setContentText("Veuillez confirmer votre choix");

ButtonType buttonType1 = new ButtonType("Choix1");
ButtonType buttonType2 = new ButtonType("Choix2");
ButtonType buttonType3 = new ButtonType("Choix3");
ButtonType buttonTypeAnnuler = new ButtonType("Annuler", ButtonData.CANCEL_CLOSE);
dialog.getButtonTypes().setAll(buttonType1, buttonType2, buttonType3, buttonTypeAnnuler);

Optional<ButtonType> choix = dialog.showAndWait();

if(choix.get() == buttonType1) {
    System.out.println("Vous avez fait le choix 1");
}
```

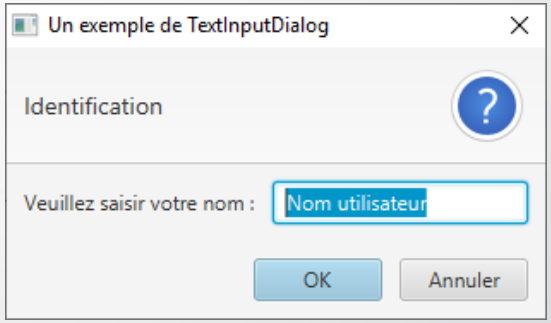


TextInputDialog

- **TextInputDialog** : permet de saisir du texte

Exemple :

```
TextInputDialog dialog = new TextInputDialog("Nom utilisateur");
dialog.setTitle("Un exemple de TextInputDialog");
dialog.setHeaderText("Identification");
dialog.setContentText("Veuillez saisir votre nom :");

Optional<String> texteSaisi = dialog.showAndWait();
// Teste si une saisie a été faite
if (texteSaisi.isPresent()) {
    System.out.println("nom saisi = "+ texteSaisi.get());
}
// autre possibilité
texteSaisi.ifPresent(new Consumer<String>() {
    @Override
    public void accept(String t) {
        System.out.println("nom saisi = "+ texteSaisi.get());
    }
});
```



On utilise la méthode isPresent ou ifPresent pour récupérer la saisie

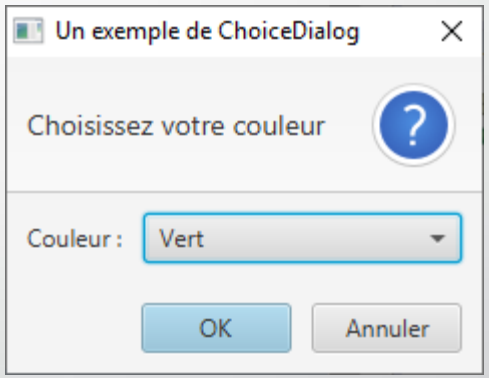
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextInputDialog.html

ChoiceDialog

- **ChoiceDialog** : permet de faire un choix dans une liste

- Exemple :

```
String[] choix = {"Bleu", "Rouge", "Vert", "Violet"};

ChoiceDialog<String> dialog = new ChoiceDialog<>(choix[2], choix);
dialog.setTitle("Un exemple de ChoiceDialog");
dialog.setHeaderText("Choisissez votre couleur");
dialog.setContentText("Couleur :");

Optional<String> selection = dialog.showAndWait();
// Teste si un choix a été faite
if (selection.isPresent()) {
    System.out.println("nom saisi = "+ selection.get());
}
```



https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ChoiceDialog.html

On peut personnaliser notre boîte de dialogue :

- définir un contenu étendu (qui peut être plié/déplié) :

```
dialog.getDialogPane().setExpandableContent(node);
```

- spécifier une nouvelle icône (par défaut l'icône est déterminée par le type de la boîte de dialogue) :

```
dialog.setGraphic(new ImageView("/icone.png"));
```

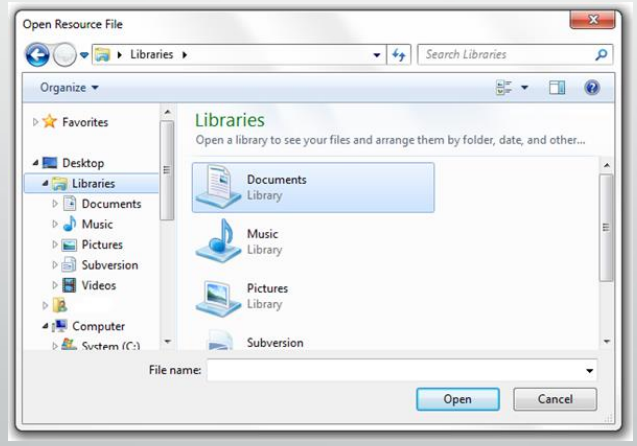
- la rendre modale ou non modale (c'est-à-dire que l'on autorise l'utilisateur à interagir avec la fenêtre parent sans devoir fermer la boite de dialogue) :

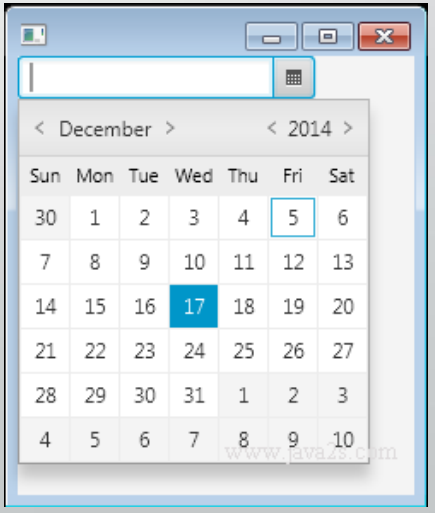
```
dialog.initModality(Modality.NONE);
```

Les boites de dialogues spécialisées

- **Le FileChooser** : permet de naviguer dans le système de fichiers afin de choisir un fichier (affichage dépend du système d'exploitation)
- **Le DirectoryChooser :** permet de naviguer dans le système de fichiers afin de choisir un répertoire
- **Le DatePicker :** permet de sélectionner une date dans un calendrier
- **Le ColorPicker :** permet de choisir une couleur dans une palette,

il peut même définir une nuance de couleur





Les propriétés et le mécanisme de binding

Le concept de propriété

- Une **propriété** (nommée aussi **attribut**) est un élément qui permet de décrire une classe

  (tout du moins une partie de cette classe)

- On lui adjoint des méthodes pour pouvoir
  - la lire : **get / accesseur**
  - la modifier **: set / mutateur**

- Normée en Java :
  - attribut ***solde*** de type **double** → ***getSolde( )*** et ***setSolde(double)***

Exemple attribut Java

```
public class CompteBancaire {
   private double solde;

   public double getSolde() {
      return solde;
   }

   public void setSolde(double solde) {
      this.solde = solde;
   }
}
```

En JavaFX, on va plus loin : on introduit un objet de type **DoubleProperty** et une méthode ***soldeProperty( )*** qui retourne cet objet

Property JavaFX

```
public class CompteBancaire {

 private DoubleProperty solde = new SimpleDoubleProperty();

   public final double getSolde() {
       return solde.get();
   }
   public final void setSolde(double solde) {
       this.solde.set(solde);
   }

   public DoubleProperty soldeProperty() {
        return solde;
   }
}
```

Propriétés JavaFX 1/2

- La classe abstraite **DoubleProperty :**
  - C'est une Property qui emballe une valeur de type double (**Wrapper**) : elle permet de **lire** et **modifier** la valeur (comme la classe Double), mais aussi **d'observer et de lier ses changements, ainsi que son état**
- La classe **SimpleDoubleProperty** :
  - C'est une implémentation, une classe concrète

Exemples de classes Wrapper associées aux Property (tableau non exhaustif)		
type	**classe wrapper abstraite**	**implémentation**
int	IntegerProperty	SimpleIntegerProperty
double	DoubleProperty	SimpleDoubleProperty
String	StringProperty	SimpleStringProperty
List	ListProperty<E>	SimpleListProperty<E>
Object	ObjectProperty<T>	SimpleObjectProperty<T>

Propriétés JavaFX 2/2

- Eléments manipulables à l'aide de getter/setter
  - **getXxx()** → getSolde() : implémente l'interface ReadOnlyProperty<T>
  - **setXxx()** → setSolde() : implémente l'interface WritableValue<T>
  - **xxxProperty() :** retourne un objet implémentant l'interface **Property**
- **Intérêts :**
  - peuvent être **observables** : déclenchent un évènement
    - lorsque leur valeur change
    - lorsqu'elles deviennent invalides
  - peuvent être liées (**bound**) entre-elles : c'est le **binding !**

Chaque composant JavaFX possède de nombreuses propriétés

Exemple avec MenuItem

- Exemple de propriétés pour le composant MenuItem :

Property Summary

All Methods | **Instance Methods** | **Concrete Methods**

Type	Property and Description
`ObjectProperty<KeyCombination>`	**`accelerator`** The accelerator property enables accessing the associated action in one keystroke.
`BooleanProperty`	**`disable`** Sets the individual disabled state of this MenuItem.
`ObjectProperty<Node>`	**`graphic`** An optional graphic for the `MenuItem`.
`StringProperty`	**`id`** The id of this MenuItem.
`BooleanProperty`	**`mnemonicParsing`** MnemonicParsing property to enable/disable text parsing.
`ObjectProperty<EventHandler<ActionEvent>>`	**`onAction`** The action, which is invoked whenever the MenuItem is fired.
`ObjectProperty<EventHandler<Event>>`	**`onMenuValidation`** The event handler that is associated with invocation of an accelerator for a MenuItem.
`ReadOnlyObjectProperty<Menu>`	**`parentMenu`** This is the **`Menu`** in which this `MenuItem` exists.
`ReadOnlyObjectProperty<ContextMenu>`	**`parentPopup`** This is the **`ContextMenu`** in which this `MenuItem` exists.
`StringProperty`	**`style`** A string representation of the CSS style associated with this specific MenuItem.
`StringProperty`	**`text`** The text to display in the `MenuItem`.
`BooleanProperty`	**`visible`** Specifies whether this MenuItem should be rendered as part of the scene graph.

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/MenuItem.html

Observation des propriétés

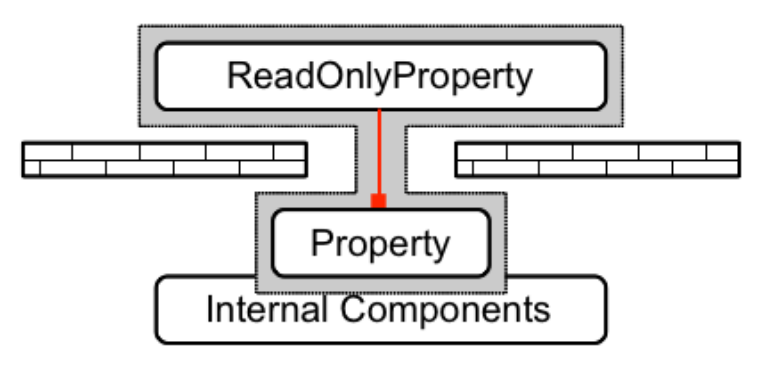
- Les Property implémentent des interfaces permettant de leur attacher des écouteurs (Listener) :
  - **ObservableValue<T> :** ***ChangeListener<T> : méthode changed()***
  - **Observable :** ***InvalidationListener<T> : méthode invalidated()***

```
solde.addListener(new ChangeListener<Number>() {
  @Override
  public void changed(
        ObservableValue<? extends Number> observable,
        Number oldValue,
        Number newValue) {
            System.out.println("Le solde a changé !!!");
        }
});
```

Property en lecture seule

- Il existe aussi des Property en lecture seule ("read-only")
- Pour qu'une Property soit en mode lecture seule, on va utiliser des classes Wrapper préfixées par "ReadOnly" et qui implémentent l'interface **ReadOnlyProperty<T>**
- Dans un tel wrapper, **il y a en réalité deux Property** :
  - une en lecture/écriture : utilisé uniquement en interne
  - une en lecture seule : c'est celle qui est communiquée à l'extérieur

Ces deux Property sont liées : la valeur wrappée est toujours synchronisée



Exemple de ReadOnlyProperty

```
private ReadOnlyDoubleWrapper solde =
                    new ReadOnlyDoubleWrapper();

public final double getSolde() {
   return solde.get();
}

public ReadOnlyDoubleProperty soldeProperty() {
   return solde.getReadOnlyProperty();
}

public final void fructifier(double taux) {
   double newSolde = solde.get() * (1 + taux);
   solde.set(newSolde);
}
```

Mise à jour possible de la valeur wrappée seulement en interne

Lier des propriétés : le binding

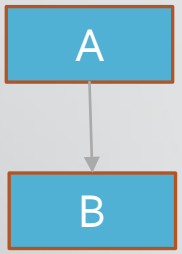
- **Possibilité de lier les propriétés entre-elles**
  - **permet de mettre à jour automatiquement une propriété en fonction d'une autre**

- **Différentes manières de lier des propriétés :**
  - **utiliser les méthodes *bind()* et *bindBidirectional()***
  - **binding de haut niveau :** fluent API et/ou classe Bindings
  - **binding de bas niveau :** classe héritant de *javafx.beans.binding.NumberBinding*

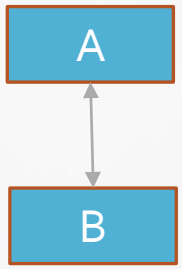
Les deux dernières manières de lier les propriétés (haut niveau et bas niveau) sont citées et illustrées avec un exemple, mais ne seront pas traitées dans le cours.

Lier des propriétés : le binding

- illustrations des liens possibles entre A et B :



B.*bind* (A)



B.*bindBidirectional* (A)
ou
A.*bindBidirectional* (B)

Liaison unidirectionnelle

(la valeur de la propriété B dépend de la valeur de la propriété A)

Liaison bidirectionnelle

(la valeur de la propriété B dépend de la valeur de la propriété A et inversement)

Pour supprimer le lien : *unbind*() et *unbindBidirectional*()

Le binding de haut niveau

- Pour des besoins de liaison plus complexes (nécessité de faire des calculs)
- Classe utilitaire Bindings

```
IntegerProperty num1 = new SimpleIntegerProperty(1);
IntegerProperty num2 = new SimpleIntegerProperty(2);
NumberBinding sum = Bindings.add(num1, num2);
System.out.println(sum.getValue());

num1.setValue(2);
System.out.println(sum.getValue());
```

- Fluent API

```
// Area = width * height
IntegerProperty width = new SimpleIntegerProperty(10);
IntegerProperty height = new SimpleIntegerProperty(10);
NumberBinding area = width.multiply(height);
```

Le binding est « lazy » evalué (calculé quand on appelle le *get()*)

Le binding de bas niveau

- Dans le cas où ce qui est fourni dans le binding de haut niveau ne suffit pas à notre besoin
- A la place, on peut alors redéfinir la méthode ***computeValue( )*** de l'une des classes de Binding : cela nous permet d'avoir plus de flexibilité
- Exemple avec la classe **DoubleBinding**

```
DoubleProperty rayon = new SimpleDoubleProperty(2);

DoubleBinding volumeSphere = new DoubleBinding() {
    {
        super.bind(rayon); // bind initial avec les paramètres en entrée
    }

    @Override
    protected double computeValue() {
        // Math.pow() élève le rayon à la puissance 3
        return (4 / 3 * Math.PI * Math.pow(rayon.get(), 3));
    }
};
```

Le binding de haut niveau et de bas niveau ne sont pas traités dans ce cours