Université de Limoges

R2.02

JavaFX™

Notions avancées (suite)

Objectifs du cours

- Rappels…
- ObservableList et ListView
- Le glisser/déposer (Drag-and-Drop)
- Apparence

Rappels

- Divers composants JavaFX :
  - Les images
  - Les dessins (Shapes, Canvas, etc)
  - Les menus
  - Les boîtes de dialogue
  - Autre composants spécialisés (ColorPicker, FileChooser, etc)
- Les propriétés en JavaFX
  - Observations des propriétés (StringProperty, ObjectProperty, etc)
  - Le mécanisme de binding

Utilisation d'une liste en JavaFX

- Nos IHM ont couramment besoin d'afficher des listes.
- Mais comment être capable de notifier les UI controls en cas d'ajout, modification ou suppression d'un élément d'une liste ?
- Synchroniser à la main les changements liés au modèle et les répercuter sur la vue est très fastidieux…
- JavaFX répond à ce besoin avec le package javafx.collections.

- On va s'intéresser à l'interface **ObservableList<T>**.
- Elle sert de modèle à de nombreux composants tels que des gestionnaires de mise en page, listes, arbres, tables graphiques, etc…
- Elle représente une liste permettant de notifier les modifications faites dans ses données.

- Il existe aussi : **ObservableArray, ObservableSet<T>** et **ObservableMap<K, V>**.

Créer une ObservableList

Pour créer une ObservableList, on peut s'aider de la classe **FXCollections :**

- propose des méthodes manipulant les Collections (wrapper plus performant).
- dispose d'une factory permettant de créer une ObservableList à partir des types de Collections classiques.

https://docs.oracle.com/javase/8/javafx/api/javafx/collections/FXCollections.html

Exemples :

```
// Exemple 1
final ObservableList<String> items =
            FXCollections.observableArrayList(
              "Single", "Double", "Suite", "Family App");

// Exemple 2
// créer une liste Java
List<String> liste = new ArrayList<>();
// ajouter des éléments
liste.add("d");
liste.add("b");
liste.add("a");
liste.add("c");
// On l'enveloppe dans une ObservableList pour pouvoir l'observer
ObservableList<String> observableList = FXCollections.observableList(liste);
```

Ecouter une ObservableList

- Utiliser l'écouteur javafx.collections.ListChangeListener<T>

```
maListe.addListener(new ListChangeListener<T>() {
    @Override
    public void onChanged(ListChangeListener.Change<? extends T> change) {
        // analyse et traitement des changements
        while (change.next()) {
            if (change.wasReplaced()) {
                // une ou plusieurs valeurs ont été remplacées
            }
            if (change.wasPermutated()) {
                // permutation entre 2 éléments de la liste
            }
            if (change.wasUpdated()) {
                // une ou plusieurs valeurs ont été modifiées
            }
            if (change.wasAdded()) {
                // une ou plusieurs valeurs ont été ajoutées
            }
            if (change.wasRemoved()) {
                // une ou plusieurs valeurs ont été supprimées
            }
        }

    }
});
```

Pour améliorer les performances et minimiser le nombre d'événements publiés, l'objet "change" reçu dans la méthode *onChanged*() d'un ListChangeListener est en fait un rapport agrégé contenant les récentes modifications sur la liste observable.

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

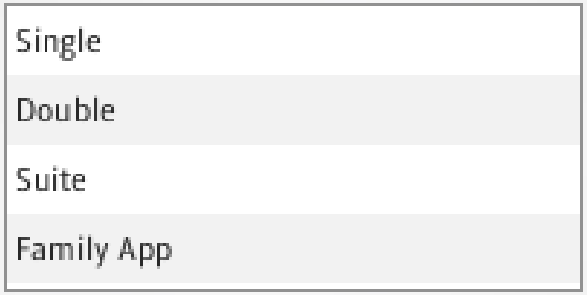
Le composant ListView

- Le composant ListView<T> permet de représenter une liste scrollable d'éléments (génériques de type T).
- Permet un affichage vertical (par défaut) ou horizontal.
- Les éléments sont contenus en interne dans la propriété *items* (**ObervableList**) → permet au composant d'être notifié si des modifications sont faites sur les données.
- Interactions possibles avec l'utilisateur :
  - Focus → focusModel
  - Sélection → selectionModel.selectionMode (SINGLE ou MULTIPLE)
  - Edition → passer par CellFactory et ListCell

https://docs.oracle.com/javafx/2/ui_controls/list-view.htm

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

- Exemple :




```
final ObservableList<String> items =
  FXCollections.observableArrayList(
   "Single", "Double", "Suite", "Family App");

final ListView<String> liste = new ListView<String>();
liste.setItems(items);
```

Le glisser/déposer (Drag-and-Drop)

- Pré-requis : la sérialisation

- Définition
- Principe en JavaFX
- Focus sur le Drag and Drop (ou DnD)
  - Initier un DnD
  - Transfert de données
  - Évènements lors d'un DnD

Prérequis : la sérialisation d'objets

Définitions :

- **Sérialisation :** écrire un objet et ses objets référencés dans un flux binaire.
- **Déserialisation :** lire un objet dans un flux binaire pour le reconstruire.

Intérêts :

- **Persistance** (sauvegarder un objet dans un fichier)
- **Distribution** (véhiculer un objet sur un réseau)

Pré-requis de la classe :

- implémenter l'interface java.io.**Serializable**
- identifiant de version → **serialVersionUID** (compatibilité entre versions)
- mot réservé **transient** → variable d'instance non sérialisée

Comment ?

- Flux (java.io) de transformation **ObjectInputStream / ObjectOutputStream**

Une classe Serializable

```java
import java.io.Serializable;

public class Joueur implements Serializable {

    private static final long serialVersionUID = 1L;

    private String nom;

    private String prenom;

    private transient String etatForme;

    public Joueur(String nom, String prenom, String etatForme) {
        this.nom = nom;
        this.prenom = prenom;
        this.etatForme = etatForme;
    }

    // getX, setX, ...

    @Override
    public String toString() {
        return "Joueur [nom=" + nom
            + ", prenom=" + prenom
            + ", etatForme=" + etatForme + "]";
    }

}
```

Implémente java.io.Serializable

identifiant de version

Variable d'instance ignorée lors de la sérialisation

Exemples de code

Sérialisation

```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class Serialisation {

    public static void main(String[] args) {

        Joueur player = new Joueur("Lucky", "Luke", "en_forme");
        System.out.println(player);

        ObjectOutputStream objectOutputStream = null;
        try {
            final FileOutputStream fichier =
                new FileOutputStream("C:/temp/joueur.ser");
            objectOutputStream = new ObjectOutputStream(fichier);
            objectOutputStream.writeObject(player);
            objectOutputStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (objectOutputStream != null) {
                try {
                    objectOutputStream.close();
                } catch (final IOException ex) {
                    ex.printStackTrace();
                }
            }
        }

    }

}
```

```
Joueur [nom=Lucky, prenom=Luke, etatForme=en_forme]
```

Désérialisation

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class Deserialisation {

    public static void main(String[] args) {

        try (ObjectInputStream objectInputStream
                = new ObjectInputStream(
                        new FileInputStream(
                                "C:/temp/joueur.ser"))) {
            Joueur player =
                    (Joueur) objectInputStream.readObject();

            System.out.println(player);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

    }

}
```

```
Joueur [nom=Lucky, prenom=Luke, etatForme=null]
```

conséquence de transient
→ valeur par défaut

Le glisser/déposer (Drag-and-Drop)

- Définition
- Principe en JavaFX
- Focus sur le Drag and Drop (ou DnD)
  - Initier un DnD
  - Transfert de données
  - Évènements lors d'un DnD

Définition

- Technique permettant à l'utilisateur de sélectionner un objet à l'écran, le déplacer jusqu'à une autre position, puis le lâcher pour déclencher une action sur cet objet.
- Manipulation pouvant être faite à partir de la souris ou du doigt (tactile).
- Procédé aujourd'hui très largement implanté dans les interfaces graphiques des mobiles et tablettes, ainsi que dans les ordinateurs. C'est un élément-clé de l'ergonomie des interfaces utilisateur modernes (simplicité du geste).
- Eléments personnalisables : le curseur de la souris, les éléments qui peuvent être déplacés, la façon dont ceux-ci sont signalés et les éléments qui peuvent servir de destination.

- Quelques usages :
  - copier / déplacer / ouvrir un élément / effacer / désinstaller / imprimer
- Applications qui prennent en charge cette technique :
  - systèmes d'exploitation, éditeurs de texte, navigateurs, etc.

Principe en JavaFX

1. Press : l'utilisateur clique avec la souris, sur un composant source (Scene, Node).
2. Drag : l'utilisateur effectue un déplacement tout en maintenant le bouton de la souris enfoncé → déclenche un évènement de type DRAG_DETECTED.
3. Release : l'utilisateur relâche le bouton de la souris.

Il existe trois types de mouvements de souris (gestures), faisant intervenir différents évènements

Mode de la gesture	Évènement concerné	Commentaires
simple press-drag-release	MouseEvent	Mouvement ne concernant qu'un seul nœud. Permet par exemple de redimensionner une forme, de la déplacer, etc. → mode déclenché par défaut
full press-drag-release	MouseDragEvent	Permet de faire glisser un nœud par dessus d'autres nœuds de l'application, tout en les notifiant (ceux parcourus pendant le déplacement de la souris). (activé en appelant la méthode ***startFullDrag***).
drag-and-drop	DragEvent	C'est le copier/coller supporté par la plate-forme. Permet de transférer des données. Fonctionne entre différentes applications (pas forcément en JavaFX). (activé via la méthode ***startDragAndDrop***).

On va s'intéresser à cette gesture

Focus sur le Drag and Drop (ou DnD)

- C'est un transfert entre 2 objets : une source et une cible.
- Cela peut se faire entre :
  - des composants d'une même application JavaFX (Node ou Scene).
  - deux applications Java distinctes : votre application JavaFX et un autre client Java (qui n'est pas forcément en JavaFX).
  - votre application JavaFX et une application tierce (native).

- Se base sur l'évènement javafx.scene.input.**DragEvent**

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

- Récupérer la source : `final Object source = dragEvent.getGestureSource();`
- Les données sont transférés via un dragboard (même principe que le presse-papier).
- Différents modes de transferts sont disponibles (copier, déplacer, etc.).
- Différents types de données peuvent être transférées, telles que : du texte, des images, des URLs, des fichiers, du contenu binaire, etc.

Initier un DnD sur une source

- La détection est basée sur la combinaison d'un clic et d'un déplacement de la souris (bouton maintenu enfoncé) : un évènement de type **DRAG_DETECTED** sera déclenché sur la source potentielle de la gesture (l'objet cliqué).
- Rattacher un Event Handler pour traiter cet évènement : il faut alors appeler la méthode ***startDragAndDrop*** (en précisant les **TransferMode** en paramètre) :
  - bascule la gesture en mode drag-and-drop
  - des évènements de type **DragEvent** seront délivrés par le système
  - retourne un objet **Dragboard** permettant d'y associer un contenu de type **ClipboardContent** (encapsule les données à transférer).

Le DnD ne démarrera pas si le contenu du Dragboard reste vide.

```
Rectangle source = new Rectangle(100, 100);
source.setOnDragDetected(new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent event) {
        Dragboard dragboard = source.startDragAndDrop(TransferMode.ANY);

        ClipboardContent content = new ClipboardContent();
        content.putString("Hello!");
        dragboard.setContent(content);

        event.consume();
    }
});
```

Les données transférées

TransferMode : durant une opération de DnD, on doit définir le mode de transfert.
Les valeurs possibles sont : **COPY, MOVE, LINK**.

- la source indique les modes de transfert supportés.
- la cible peut accepter un ou plusieurs mode(s) de transfert.

Dans une opération de DnD donnée, c'est la cible potentielle qui détermine un mode de transfert compatible avec la source.

Pendant le déplacement de la souris : l'utilisateur a une indication visuelle lui permettant de repérer les composants cibles qui acceptent le transfert.

ClipboardContent permet d'encapsuler les données à transférer.

- déposer une donnée : **put*XX***()
- vérifier la présence d'une donnée : **has*XX***()
- récupérer une donnée : **get*XX***()

(*XX* pouvant être : files, Image, String, html, rtf, url, etc)

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

Récap des évènements lors d'un DnD

- Quand un Drag-and-Drop est initié, n'importe quel objet (Node, Scene) parcouru par la souris, devient une cible potentielle. Différents évènements entrent en jeu :

- Evènements déclenchés sur les cibles potentielles :
  - **DRAG_ENTERED** et **DRAG_EXITED** : entrée et sortie du curseur de la souris dans la zone du composant (même principe que MOUSE_ENTERED et MOUSE_EXITED).
  - **DRAG_ENTERED_TARGET** et **DRAG_EXITED_TARGET** : variantes déclenchées dans la phase de bubbling.
  - **DRAG_OVER** : le curseur de la souris se trouve au-dessus de la zone du composant. Pour finaliser le DnD, la cible doit indiquer les modes de transfert supportés en appelant la méthode event.acceptTransferModes(TransferMode…).
  - **DRAG_DROPPED** : le bouton de la souris est relâché au-dessus de ce composant.

- Evènement déclenché sur la source :
  - **DRAG_DONE** : la source est notifiée du fait que les données ont été déposées sur une cible (fait suite à un évènement DRAG_DROPPED sur la cible).

Indiquer visuellement les cibles

Une cible peut adapter son apparence pour indiquer à l'utilisateur qu'il accepte les données transférées (se baser sur DRAG ENTERED/EXITED ou bien sur DRAG_OVER).

Avant d'indiquer que l'on accepte le transfert, il faut vérifier que le type de données présent dans le Dragboard est compatible. On y accède via la méthode *event.getDragboard( )*.

Exemple :

DRAG_ENTERED

```
// souris entrée dans la zone d'une cible potentielle
target.setOnDragEntered(new EventHandler<DragEvent>() {
    @Override
    public void handle(DragEvent event) {
        // verifier que le dragboard contient
        // des données au format attendu
        if (event.getDragboard().hasString()) {
            // change la couleur en rouge
            target.setFill(Color.RED);
        }

        event.consume();
    }
});
```

DRAG_EXITED

```
// souris sortie de la zone d'une cible potentielle
target.setOnDragExited(
    new EventHandler<DragEvent>() {

    @Override
    public void handle(DragEvent event) {
        // change la couleur en noir
        target.setFill(Color.BLACK);

        event.consume();
    }
});
```

Over : cibles survolées

- L'évènement **DRAG_OVER** donne la possibilité au composant survolé de faire savoir que c'est une cible valide. Pour cela, il faut :
  - vérifier que le type de données présent dans le Dragboard est compatible.
  - si c'est le cas, appeler la méthode **evt.*acceptTransferModes***(*TransferMode…*) en passant en paramètre, les modes de transfert supportés.
- Pour vérifier les modes de transfert, et décider, on dispose de 2 méthodes :

  Dragboard.*getTransferModes( )* : l'ensemble des modes de transfert indiqués par la source.

  Dragboard.*getTransferMode( )* : mode de transfert par défaut (issu de la plate-forme).
- Le composant n'est pas considéré comme une cible valide pour ce DnD : si la méthode n'est pas appelée pendant la propagation de l'évènement DRAG_OVER, ou bien si aucun mode de transfert indiqué n'est compatible avec la source.

```
target.setOnDragOver(new EventHandler<DragEvent>() {
    @Override
    public void handle(DragEvent event) {
        Dragboard db = event.getDragboard();
        // on accepte que les donnees de type String
        if (db.hasString()) {
            // on indique nos 2 mode possibles : COPY ou MOVE
            event.acceptTransferModes(TransferMode.COPY_OR_MOVE);
        }
        event.consume();
    }
});
```

Dropped : récupérer les données

Quand on relâche le bouton de la souris au-dessus d'un composant ayant accepté le transfert avec un mode compatible avec la source (dans l'évènement DRAG_OVER) :

L'évènement **DRAG_DROPPED** est déclenché sur la cible. Dans le handler :

- possibilité de récupérer les données transférées (via le Dragboard).
- appeler la méthode ***setDropCompleted****(boolean)* pour compléter le DnD.
  Le booléen indique si les données ont été transférées avec succès ou pas.

Si la méthode n'est pas appelée, le Drag-and-Drop sera considéré comme infructueux.

```
target.setOnDragDropped(new EventHandler<DragEvent>() {
    @Override
    public void handle(DragEvent event) {
        Dragboard db = event.getDragboard();
        boolean success = false;
        // verifier que le dragboard contient une String
        if (db.hasString()) {
            // recuperer les donnees db.getString();
            //TODO exploiter les donnees recuperees...
            success = true;
        }
        // informer la source que les donnees ont bien ete recues
        event.setDropCompleted(success);

        event.consume();
    }
});
```

Done : finaliser le DnD

Une fois que le DnD est fini (quelque soit l'issue du transfert, succès ou échec, ou bien abandon), l'évènement **DRAG_DONE** est envoyé vers la source.

Dans le handler : la méthode event.***getTransferMode()*** indique à la source comment le transfert de données s'est déroulé :

- Si le mode de transfert est MOVE : cela permet à la source d'effacer ses données.
- Si le mode de transfert est NULL : cela signifie que le DnD s'est terminé sans que les données n'aient été transférées.

```
final Text src = new Text(50, 100, "DRAG ME");
src.setOnDragDone(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
        // Le drag and drop est finit
        // Si les données ont été déplacées avec succès, on les efface
        if (event.getTransferMode() == TransferMode.MOVE) {
            src.setText("");
        }
        event.consume();
    }
});
```

Transférer des données personnalisées

On peut aussi transférer des données personnalisées lors d'un Drag-and-Drop.

Il faut passer par un objet de type DataFormat.

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

- public *DataFormat*(*String*... ids) : le constructeur prend en paramètre une ou plusieurs chaîne(s) de caractères : ce sont les types MIME (Multipurpose Internet Mail Extensions) correspondant au format de données.

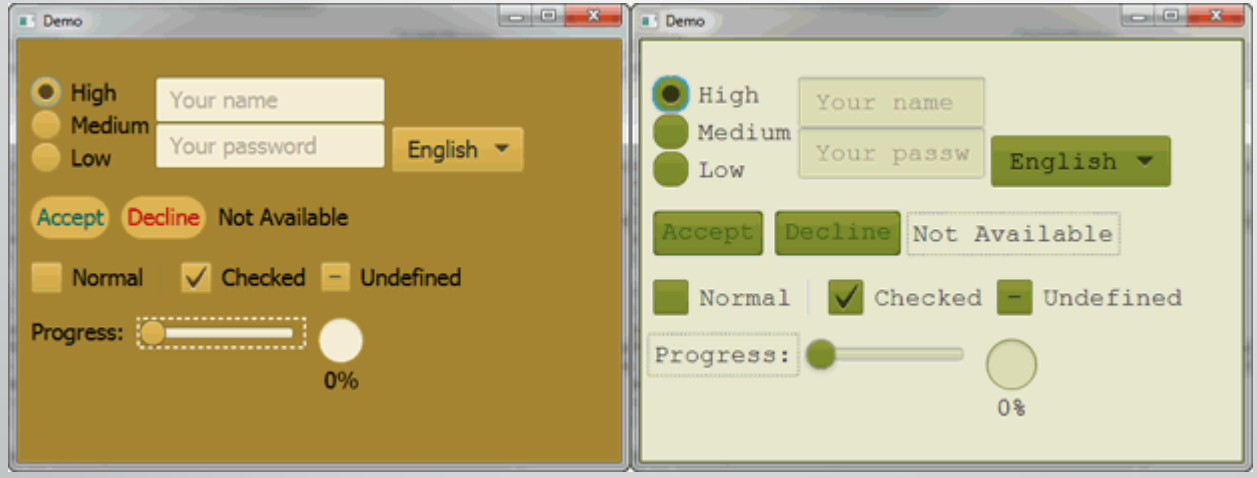
  (mais dans les faits, cela peut être n'importe quelle chaîne de caractères)

- L'objet transférés doit être ***Serializable***.

Exemple :

```
DataFormat fmt = new DataFormat("text/foo", "text/bar");
Clipboard clipboard = Clipboard.getSystemClipboard();
ClipboardContent content = new ClipboardContent();
content.put(fmt, "Hello");
clipboard.setContent(content);
```

Apparence

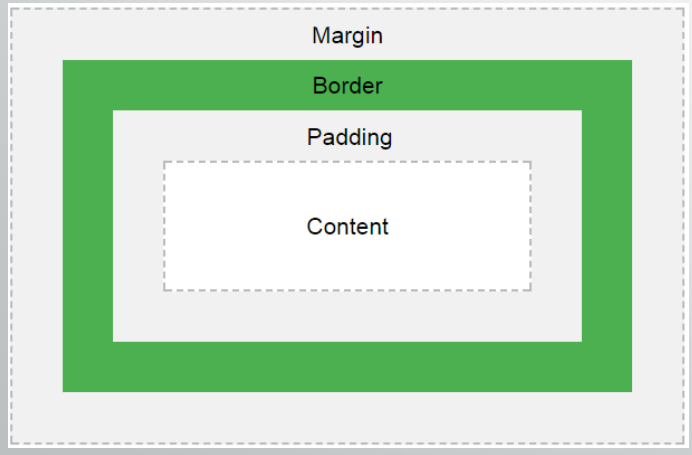

Demo
High
Medium
Low
Your name
Your password
English
Accept
Decline
Not Available
Normal
Checked
Undefined
Progress:
0%
Demo
High
Medium
Low
Your name
Your passw
English
Accept
Decline
Not Available
Normal
Checked
Undefined
Progress:
0%

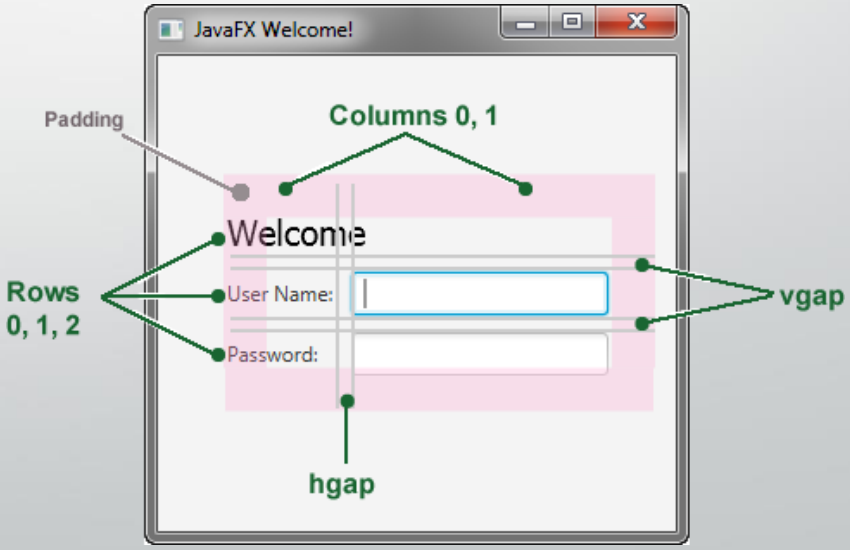
Les feuilles de style CSS

- JavaFX reprend l'utilisation des feuilles de style en cascade ou **CSS (Cascading Style Sheet)**
- C'est un standard définit par le **W3C (World Web Consortium)** permettant de mettre en forme des documents HTML (actuellement on utilise la spécification CSS 3)
- **Séparation entre la structure (html) et la présentation (css)** on peut changer radicalement l'aspect d'une page sans en modifier le contenu (géré dans des fichiers séparés)
- On peut utiliser des **sélecteurs** pour désigner des éléments
- Le rendu d'un document stylé est déterminé par les concepts de boîte et de flux CSS
- Les caractéristiques applicables aux boîtes CSS sont exprimées sous forme de couples ***"propriété: valeur;"***

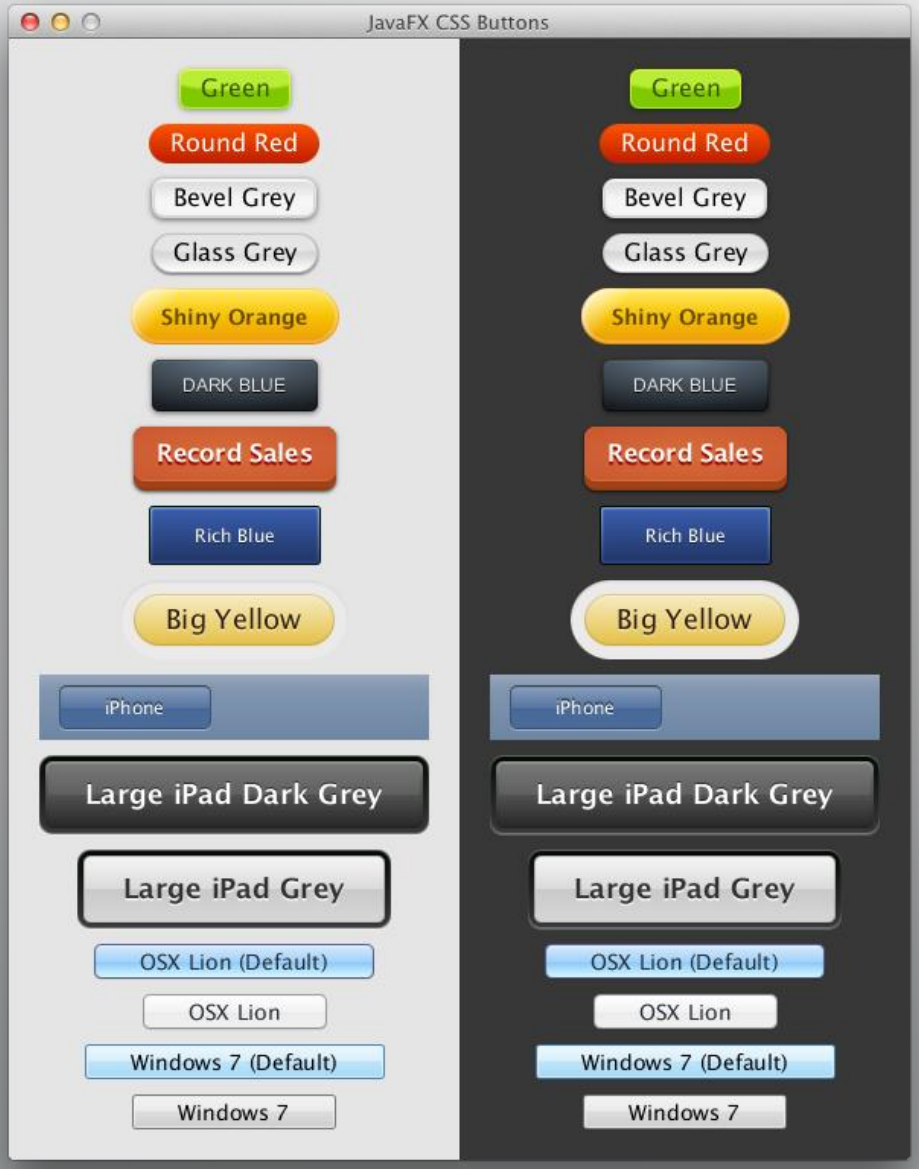
La classe Region

- Classe parente des Controls et des Panes
- Permet de définir les notions de *Margin*, *Border*, *Padding*, *Insets*, *Content*, *etc*
- Zones basées sur la spécification du Box-Model CSS3 (selon la normalisation du W3C)





Les styles avec CSS



Sélecteurs CSS

- Forme d'un sélecteur CSS

```
. ou #<nom-selecteur> <pattern> {
    -fx-<une-propriete> : <une-valeur>;
}
```

- Sélecteur par classe CSS (**.**) ou par id (**#**)

- Exemples de patterns :

```
.hbox > .button { -fx-text-fill: black; }
.label, .text { -fx-font-size: 20px; }
.num-button:hover { -fx-background-color: black; }
```

- Consulter le guide de référence JavaFX

http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

- Il existe des classes css par défaut (*.root*, *.grid-pane*, *.button*)
- Les propriétés CSS pré-définies sont préfixées par **"-fx-"**

```
.button{
    -fx-text-fill: brighter-sky-blue;
    -fx-border-color: rgba(255, 255, 255, .80);
    -fx-border-radius: 8;
    -fx-padding: 6 6 6 6;
    -fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

- Pour rajouter un fichier CSS (au niveau de la Scène) :

```
scene.getStylesheets().add("path/stylesheet.css");
// ou bien
scene.getStylesheets()
    .add(getClass().getResource("stylesheet.css")
    .toExternalForm());
```

- On peut aussi rajouter des polices de caractères (fonts)

```
Font.loadFont(getClass().getResourceAsStream("Roboto-Thin.ttf"), 10).getName();
Font.loadFont(getClass().getResourceAsStream("Roboto-Light.ttf"), 10).getName();
```

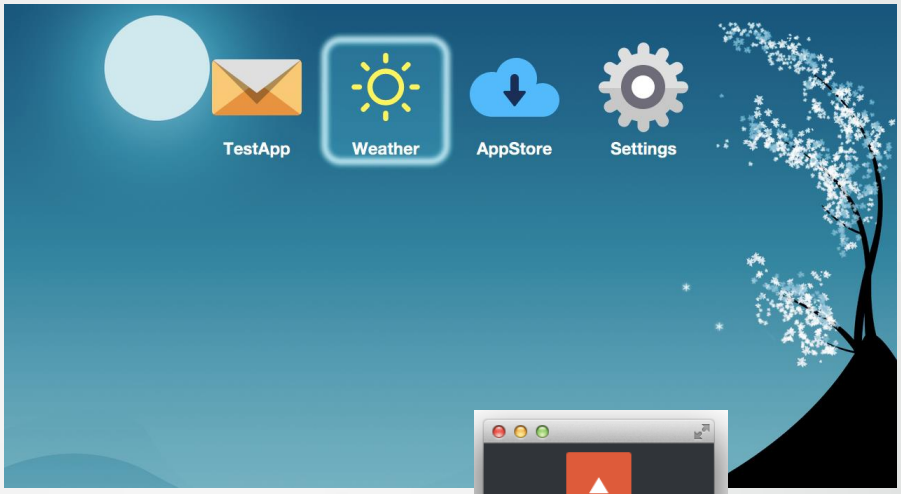
Style inline : prioritaire

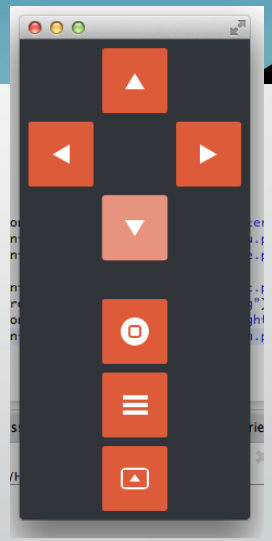
- **Modifier le style de manière programmatique**
  - Rajouter ou supprimer une classe css avec la méthode *getStyleClass().add()* ou *getStyleClass().remove()*
  - Rajouter un id de classe : *setId("mon-bouton")*, Récupérer le nœud : *Scene.lookup("#mon-bouton")*
  - Surcharger un style avec la méthode : *setStyle()*
- **Un exemple commenté :**

```
Button btn1 = new Button();
btn1.setText("click button");
// ajout d'une classe CSS
btn1.getStyleClass().add("my-default-style");
// ajout de styles CSS inline
btn1.setStyle("-fx-font-size: 30px; -fx-text-fill: green");
// remise à zéro des styles CSS
btn1.setStyle("");
```

Les Thèmes 1/2

- Deux thèmes fournis :
  - Caspian
  - Modena (défaut)





- Autres thèmes disponibles :
  - AquaFX (style MacOS)
  - Jmetro (style Windows 8) / AeroFX (Windows)
  - Flatter (style dédié aux les appareils tactiles)
  - BoxFX (lanceur d'applications sous Raspberry Pi)

Les Thèmes 2/2

- **Deux méthodes (une globale et une locale à la Scène)**
  - *setUserAgentStylesheet(String url)*

```
Application.setUserAgentStylesheet(null);

Application.setUserAgentStylesheet(STYLESHEET_CASPIAN);

Application.setUserAgentStylesheet(STYLESHEET_MODENA);
```

  - *vu précédemment :*

    scene.getStylesheets().clear()

    scene.getStylesheets().add(String url)

```
scene.getStylesheets().add(getClass().getResource("fichier.css").toExternalForm());
```