DÉVELOPPEMENT D'APPLICATIONS AVEC IHM (JAVAFX)

INTRODUCTION - CONTRÔLES, CONTENEURS, EVÈNEMENTS

1

OBJECTIFS :

- Apprendre à utiliser une librairie de création des IHM (JavaFX)
  - utilisation des composants graphiques
- Appliquer les concepts de l'O.O. à la création des IHM
  - séparation entre l'interface utilisateur et couche métier
  - programmation événementielle
  - Databinding
- Sensibilisation à l'ergonomie

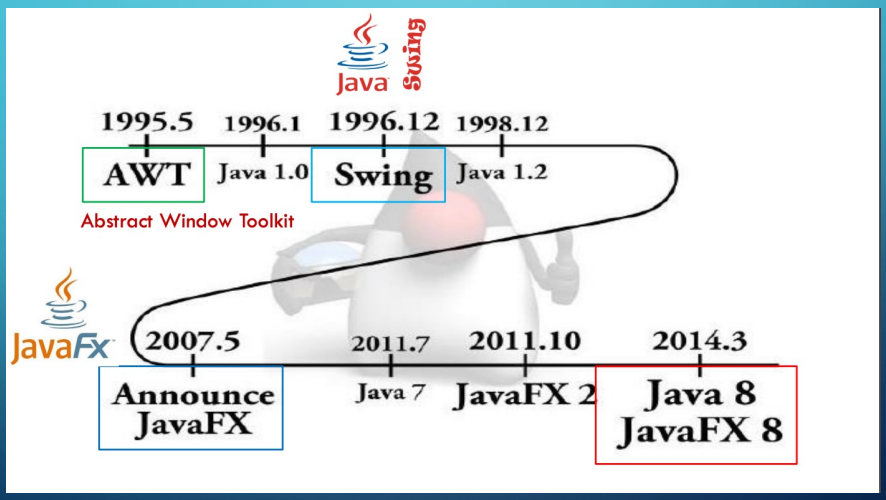
2

LES OUTILS & LA DOC :

- Java et Java FX
- IDE : Eclipse, Intelij ou autre
- Quelques livres :
  - Pro JavaFX 8 - A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients James Weaver, Weiqi Gao, Stephen Chin, Dean Iverson, Johan Vos, Adrian Chin Apress, 2014 ISBN: 978-1430265740
  - JavaFX 8 - Introduction by Example Carl Dea et Mark Heckler Apress, 2014 ISBN: 978-1430264606
  - Mastering JavaFX 8 Controls Hendrik Ebbers McGraw-Hill Professional - Oracle Press, 2014 ISBN: 978-0071833776

3

HISTORIQUE :



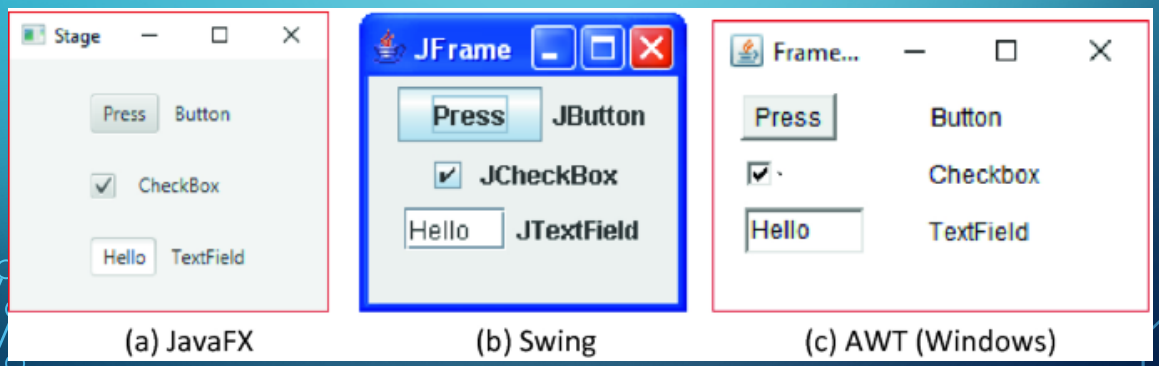
4

HISTORIQUE :

- A l'origine du langage Java , les interfaces graphiques étaient créées en utilisant la librairie AWT (java.awt)
  - Composants "lourds" (heavyweight)
  - Difficulté de créer des applications multiplateformes
- Rapidement, la librairie Swing (javax.swing) est venu compléter (et partiellement remplacer) la librairie AWT
  - Composants "légers"(lightweight) dessinés par la librairie;
  - Tout les composants de Swing exceptés JApplet, JDialog, JFrame, Jwindow sont des composants légers
  - Offre plus de composants
  - Créer des applications multiplateformes
- JavaFX 1 a tenté, sans grand succès, de remplacer Swing et Créer des interfaces graphiques pour toutes les sortes d'applications (mobiles, web, sur poste de travail, etc);

5

HISTORIQUE :



(a) JavaFX (b) Swing (c) AWT (Windows)

6

JAVA VS JAVAFX :

- Java est une plateforme
  - langage de programmation orienté objet de haut niveau
  - moteur d'exécution (JVM)
  - interface de programmation d'application standardisée (API)
- JavaFX est un framework Java pour développer des IHM
  - successeur de Swing
  - créé et développé par Sun → Oracle → OpenJFX
  - le cycle des versions JavaFX est calqué sur celui de Java

7

JAVA VS JAVAFX :

- JavaFX est un framework Java pour développer des IHM
  - caractéristiques :
    - une riche libraire de composants
    - possibilité de décrire l'interface dans un format simplifié (XML)
    - génération des IHM via un outil interactif (Scene Builder)
    - une séparation claire entre la vue utilisateur et le code métier

8

JAVA VS JAVAFX :

- JavaFX étant le résultat de développements récents, il bénéficie de concepts modernes qui en font un framework intéressant pour la réalisation d'applications dans des domaines très divers.
- JavaFX est très bien doté pour développer des interfaces riches en relation avec des données stockées dans des bases de données ou accessibles au travers de serveurs d'informations.
- Sa riche librairie graphique 2D et 3D lui donne également un intéressant potentiel dans des domaines variés :
  - Représentations graphiques
  - Animations graphiques
  - Applications multimédia
  - Jeux

9

JAVAFX : PRÉSENTATION

- JavaFX est une bibliothèque riche permettant d'expérimenter et d'apprendre de nombreux mécanismes que l'on retrouve dans la plupart des boites à outils pour faire des IHM :
  - Les widgets (composants graphiques interactifs, comme un bouton)
  - Les conteneurs qui permettent de construire et structurer une IHM
  - Les notions de Fenêtres, pointeurs, menus et la plupart des éléments habituels
  - La programmation par évènements qui permet à une application de réagir aux actions d'un utilisateur

10

JAVA VS JAVAFX :

La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces (I/F) graphiques des applications :

- Manière déclarative
- Manière procédurale

11

JAVA VS JAVAFX :

La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces (I/F) graphiques des applications :

- Manière déclarative
  - En décrivant l'interface dans un fichier FXML (syntaxe XML)
  - L'utilitaire graphique Scene Builder facilite la création et la gestion des fichiers FXML
  - L'interface peut être créée par un designer (sans connaissance Java, ou presque...)
  - Séparation entre présentation et logique de l'application (MVC)

12

JAVA VS JAVAFX :

La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces (I/F) graphiques des applications :

- Manière procédurale
  - Utilisation d'API pour construire l'interface avec du code Java
  - Création et manipulation dynamique des interfaces
  - Création d'extensions et variantes (par héritage)
  - Homogénéité des sources de l'application
- Il est possible de mélanger les deux techniques au sein d'une même application (l'API javafx.fxml permet de faire le lien entre les deux).

13

JAVAFX : DÉPLOIEMENT

Une application JavaFX peut être déployée (mise à disposition des utilisateurs) de différentes manières :

- Installée localement comme une application autonome (standalone/desktop application)
  - Semblable à une application native
  - La machine virtuelle Java peut être intégrée ou non dans l'exécutable (.exe ou .jar)
- Installée sur un serveur et intégrée dans une page web
  - Lancée depuis un navigateur, en cliquant sur un lien ou sur un autre élément actif de la page
  - Lancée automatiquement dès qu'une page est chargée
  - Utilise la technique Java Web Start (fichier JNLP + descripteur de déploiement XML)
  - Une fois téléchargée, l'application peut également être lancée horsconnexion (mise en cache local)

14

JAVAFX : PREMIÈRE VUE

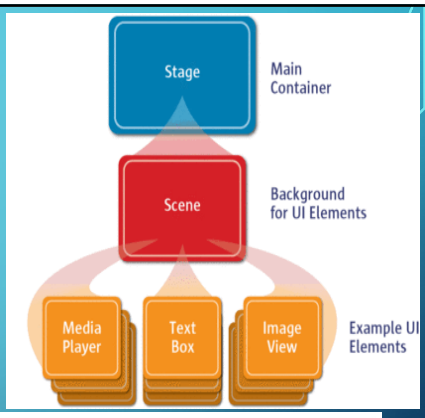
Le point d'entrée pour les applications JavaFX est la classe Application.

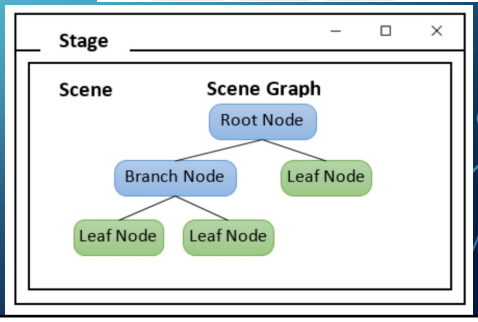
- JavaFX exécute dans l'ordre, chaque fois qu'une application est lancée, les actions suivantes :
  - Construction d'une instance de la classe Application spécifiée
  - Appel de la méthode init ()
  - Appel de la méthode start (javafx.stage.Stage)
    - le point d'entrée de l'application
  - Attente de l'achèvement de l'application,
  - Appel de la méthode stop ()

15

JAVAFX : PREMIÈRE VUE

- L'application est codée en créant une sous-classe de Application.
- La fenêtre principale d'une application est représentée par un objet de type Stage qui est fourni par le système au lancement de l'application.
- L'interface est représentée par un objet de type Scene qu'il faut créer et associer à la fenêtre (Stage).
- La scène est composée des différents éléments de l'interface graphique (composants de l'interface graphique) qui sont des objets de type Node.
- La méthode start() construit le tout.





16

JAVAFX : CYCLE DE VIE

Le point d'entrée d'une application JavaFX est constitué de l'instance de la classe Application (généralement une sous-classe).

Lors du lancement d'une application par la méthode statique Application.launch() le runtime JavaFX effectue les opérations suivantes :

1. Crée une instance de la classe qui hérite de Application
2. Appelle la méthode init()
3. Appelle la méthode start() et lui passe en paramètre une instance de Stage (qui représente la fenêtre principale [primary stage])
4. Attend ensuite que l'application se termine; cela se produit lorsque :
   - La dernière fenêtre de l'application a été fermée (et Platform.isImplicitExit() retourne true)
   - L'application appelle Platform.exit() (ne pas utiliser System.Exit())
5. Appelle la méthode stop()

17

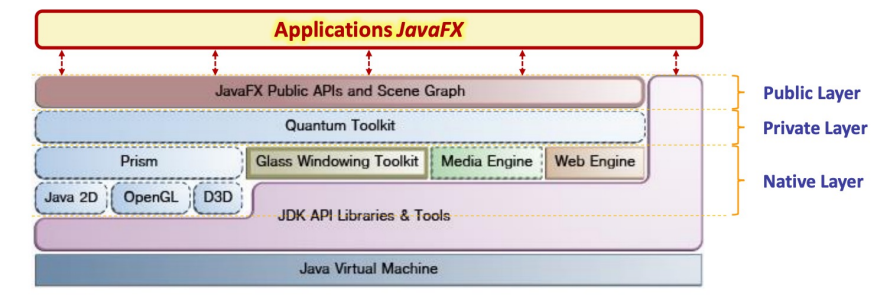
JAVAFX : CYCLE DE VIE

- La méthode launch() est généralement lancée depuis la méthode main(). Elle est implicitement lancée s'il n'y a pas de méthode main() (ce qui est toléré depuis Java 8).
- Des paramètres de lancement peuvent être récupérés en invoquant la méthode getParameters() dans la méthode init() ou ultérieurement.
- La méthode start() est abstraite et doit être redéfinie.
- Les méthodes init() et stop() ne doivent pas obligatoirement être redéfinies (par défaut elle ne font rien).
- La méthode start() s'exécute dans le JavaFX Application Thread. C'est dans ce thread que doit être construite l'interface et que doivent être exécutées toutes les opérations qui agissent sur des composants attachés à une scène (live components).

18

JAVAFX : ARCHITECTURE

L'architecture technique de la plateforme JavaFX est composée de plusieurs couches (library stack) qui reposent sur la machine virtuelle Java (JVM).



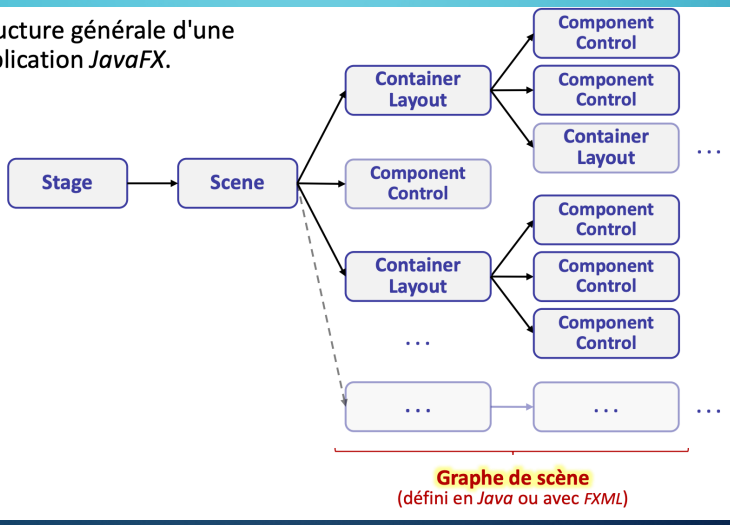
Les développeurs ne devrait utiliser que la couche (API) publique car les couches inférieures sont susceptibles de subir des changements importants au fil des versions (sans aucune garantie de compatibilité).

19

JAVAFX : STRUCTURE

Structure générale d'une application JavaFX

- Structure générale d'une application *JavaFX*.

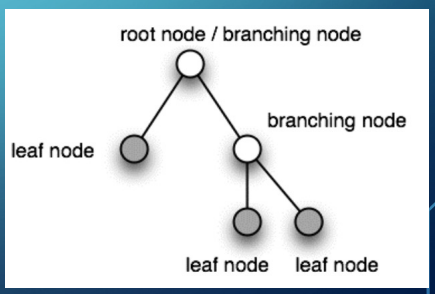


20

JAVAFX : STRUCTURE

Graphe de scène : (scene graph) est une notion importante qui représente la structure hiérarchique de l'interface graphique.

- Techniquement, c'est un graphe acyclique orienté (arbre orienté) avec :
  - une racine (root)
  - des nœuds (nodes)
  - des arcs qui représentent les relations parent-enfant
- Les nœuds (nodes) peuvent être de trois types :
  - Racine
  - Nœud intermédiaire
  - Feuille (leaf)

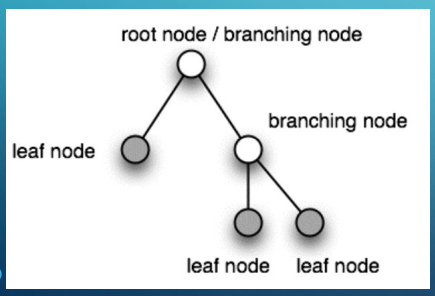


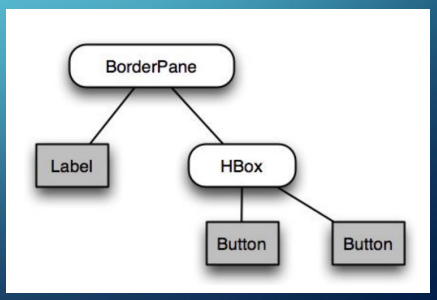
21

JAVAFX : STRUCTURE

Graphe de scène : (scene graph)

Les feuilles de l'arbre sont généralement constitués de composants visibles (boutons, champs texte, …) et les nœuds intermédiaires (y compris la racine) sont généralement des éléments de structuration (souvent invisibles), typiquement des conteneurs ou panneaux de différents types (HBox, VBox, BorderPane, …).





22

JAVAFX : STRUCTURE

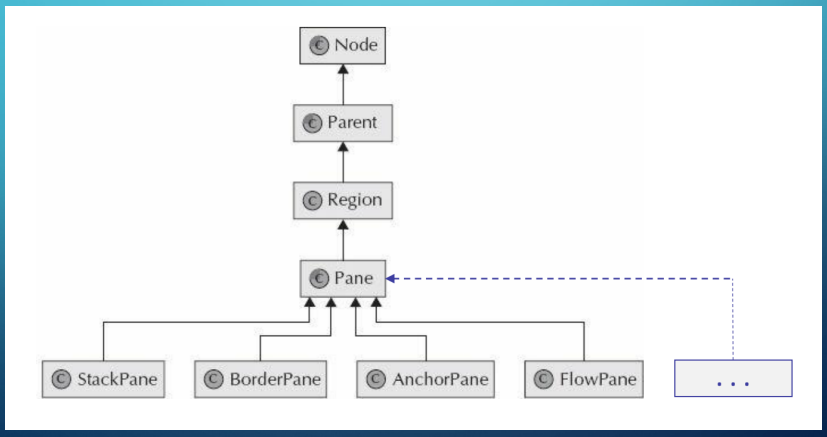
Parmi les sous-classes de Node on distingue différentes familles :

- Les formes primitives (Shape) 2D et 3D
  - Line, Circle, Rectangle, Box, Cylinder, …
- Les conteneurs (Layout-Pane) qui se chargent de la disposition (layout) des composants enfants et qui ont comme classe parente Pane.
  - AnchorPane, BorderPane, GridPane, HBox, VBox, …
- Les composants standard (Controls) qui étendent la classe Control.
  - Label, Button, TextField, ComboBox, …
- Les composants spécialisés qui sont dédiés à un domaine particulier (par exemple : lecteur multimédia, navigateur web, etc.).
  - MediaView, WebView, ImageView, Canvas, Chart, …

23

JAVAFX : CONTENEUR

Les conteneurs (Layout-Pane) représentent une famille importante parmi les sous-classes de Node. Ils ont pour classe parente Pane et Region qui possèdent de nombreuses propriétés et méthodes héritées par tous les conteneurs.



24

JAVAFX : CONTENEUR

Les conteneurs (Layout-Pane) représentent une famille importante parmi les sous-classes de Node. Ils ont pour classe parente Pane et Region qui possèdent de nombreuses propriétés et méthodes héritées par tous les conteneurs.

La classe Region est la classe parente des composants (Controls) et des conteneurs (Layout-Panes).

- Elle définit des propriétés qui affectent la représentation visuelle.
- Les différentes zones d'une région sont basées sur la spécification du Box-Model CSS3 (selon normalisation du W3C).
- Elles définissent les notions Margin, Border, Padding, Insets, Content

25

JAVAFX : CONTENEUR

Les conteneurs (Layout-Pane) représentent une famille importante parmi les sous-classes de Node. Ils ont pour classe parente Pane et Region qui possèdent de nombreuses propriétés et méthodes héritées par tous les conteneurs.

La classe Region est la classe parente des composants (Controls) et des conteneurs (Layout-Panes).

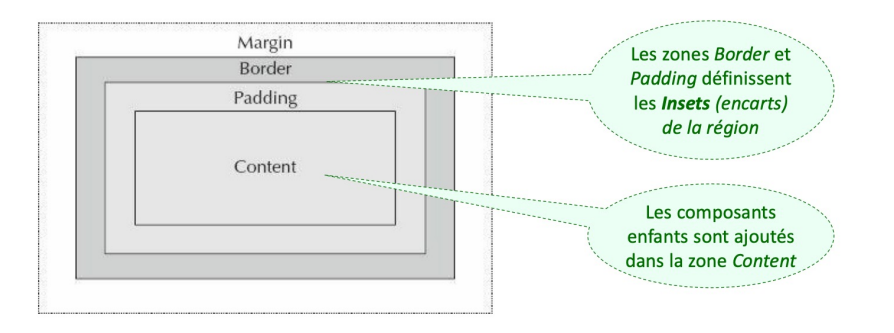
- Elle définit des propriétés qui affectent la représentation visuelle.
- Les différentes zones d'une région sont basées sur la spécification du Box-Model CSS3 (selon normalisation du W3C).
- Elles définissent les notions Margin, Border, Padding, Insets, Content

26

JAVAFX : CONTENEUR

La classe Region est la classe parente des composants (Controls) et des conteneurs (Layout-Panes).

- Elles définissent les notions Margin, Border, Padding, Insets, Content



27

JAVAFX : LOOK AND FEEL

La notion de style, skin, thème ou look and feel (L&F) caractérise l'ensemble des aspects visuels de l'interface graphique et de ses composants (forme, couleur, texture, ombre, police de caractères, …). ☐ En JavaFX, le style des composants est défini par des feuilles de style de type CSS. Il est ainsi possible de changer globalement l'aspect de l'interface sans avoir à modifier le code de l'application. ☐ La méthode setUserAgentStylesheet() de la classe Application permet d'indiquer l'URL de la feuille de style qui est à appliquer globalement. ☐ Deux styles, nommés Modena et Caspian, sont prédéfinis et sont associés aux constantes : • STYLESHEET_MODENA : Utilisé par défaut depuis JavaFX 8 • STYLESHEET_CASPIAN : A été défini pour JavaFX 2

28

JAVAFX : LOOK AND FEEL

Le style utilisé par défaut peut évoluer au fil des versions de JavaFX.

Si l'on veut fixer ou changer le look and feel des interfaces, on peut le faire au démarrage de l'application :

```
@Override public void start(Stage primaryStage) { . . .
setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

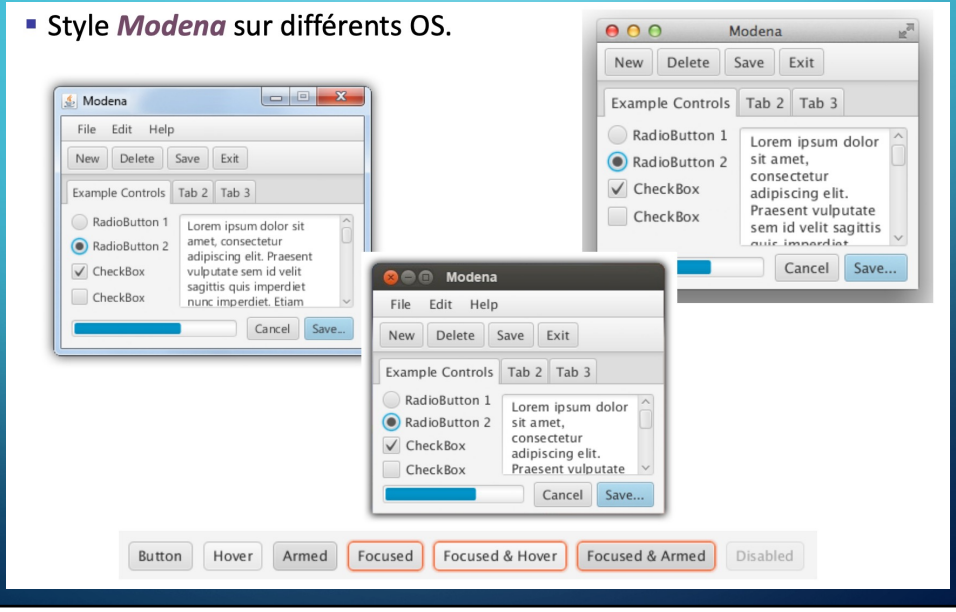
Des styles externes (third-party) peuvent également être importés et appliqués. On trouve différentes réalisations, par exemple :

- Apple Aqua (AquaFX)
- Microsoft Modern UI (JMetro) Windows-7 Aero (AeroFX)
- Twitter Bootstrap (Fextile)

29

JAVAFX : LOOK AND FEEL

- Style ***Modena*** sur différents OS.



30

JAVAFX : CONTENEUR (LAYOUT-PANE)

- La qualité d'une interface graphique repose sur de nombreux éléments mais la disposition des composants dans la fenêtre figure certainement parmi les plus importants.
- Quand on parle de la disposition (layout) d'une interface, on s'intéresse plus particulièrement :

31

JAVAFX : CONTENEUR (LAYOUT-PANE)

- La qualité d'une interface graphique repose sur de nombreux éléments mais la disposition des composants dans la fenêtre figure certainement parmi les plus importants.
- Quand on parle de la disposition (layout) d'une interface, on s'intéresse plus particulièrement :
  - A la taille des composants
  - A la position des composants
    - Position dans la fenêtre
    - Position relative des éléments les uns par rapport aux autres
  - Aux alignements et espacements qui favorisent la structuration visuelle
  - Aux bordures et aux marges (notamment autour des conteneurs)
  - Au comportement dynamique de l'interface lorsqu'on redimensionne la fenêtre, lorsqu'on déplace une barre de division (splitpane), etc.

32

JAVAFX : CONTENEUR (LAYOUT-PANE)

- Il est possible de dimensionner et de positionner les composants de manière absolue (en pixels, mm, etc.).
- Une disposition avec des valeurs absolues peut être utile et même nécessaire dans certaines situations particulières. Cependant, dans la plupart des cas, elle présente de nombreux défauts, car :
  - La taille naturelle des composants peut varier, en fonction
    - De la langue choisie (libellés, boutons, menus, …)
    - De la configuration de la machine cible (paramètres de l'OS)
    - Du look&feel, thème, skin, style (CSS) choisi par l'utilisateur
  - La taille de la fenêtre peut également varier
    - Par le souhait de l'utilisateur
    - Par obligation, pour s'adapter à la résolution de l'écran de la machine cible (pour afficher l'intégralité de l'interface)

33

JAVAFX : CONTENEUR (LAYOUT-PANE)

- Pour éviter ces inconvénients on préfère déléguer la disposition des composants à des gestionnaires de disposition (layout managers) qui sont associés à des conteneurs.
- L'idée principale est de définir des règles de disposition (des contraintes) que le gestionnaire se chargera de faire respecter en fonction du contexte spécifique de la machine cible.

34

JAVAFX : CONTENEUR (LAYOUT-PANE)

- Avec JavaFX, les layout managers sont intégrés aux conteneurs (layout-panes) et ne sont pas manipulés directement par les programmeurs (seulement au travers des propriétés des conteneurs).
- C'est donc par le choix du layout-pane et en fonction des contraintes données que sont déterminées les règles de disposition.

35

JAVAFX : CONTENEUR (LAYOUT-PANE)

- HBOX
- VBOX
- FlowPane
- TilePane
- BorderPane
- AnchorPane
- StackPane
- GridPane

36

JAVAFX : COMPONENT

- Les composants (nodes) que l'on peut placer dans des conteneurs possèdent des propriétés qui peuvent être prises en compte lors du calcul de leur disposition.
  - minWidth : Largeur minimale souhaitée pour le composant
  - prefWidth : Largeur préférée (idéale) du composant
  - maxWidth : Largeur maximale souhaitée pour le composant
  - minHeight : Hauteur minimale souhaitée pour le composant
  - prefHeight : Hauteur préférée (idéale) du composant
  - maxHeight : Hauteur maximale souhaitée pour le composant
- L'effet de ces propriétés dépend naturellement du type de conteneur (layout-pane) utilisé et de ses règles spécifiques de positionnement et de dimensionnement.

37