

Composants et layouts

INTRODUCTION AUX COMPOSANTS

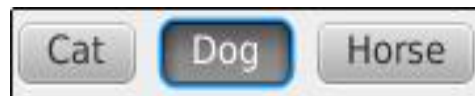
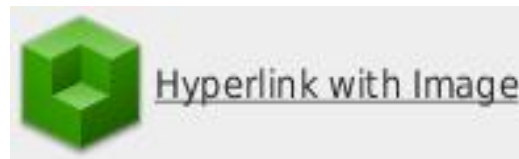
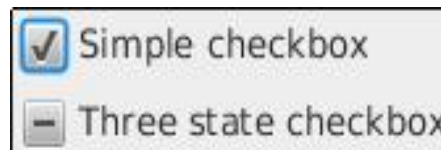
L'architecture choisie par JavaFX

- Tous les composants et conteneurs affichés dans la scène sont des Nodes
- Tous les composants sont dessinés par JavaFX
 - Pas de composants système comme en AWT ou SWT / Eclipse RCP
 - Il n'est pour l'instant pas possible d'accéder aux API de dessin très bas niveau (instructions OpenGL notamment)
 - Tout est vectorisé, y compris les textes!
- Ils sont décorés à l'aide de feuille de style CSS / skins

LES COMPOSANTS DE JAVAFX

Les basiques

- Button
- CheckBox
- Hyperlink
- RadioButton
- ToggleButton

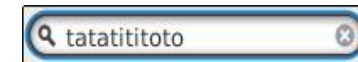


LES COMPOSANTS DE JAVAFX

Affichage et édition de textes

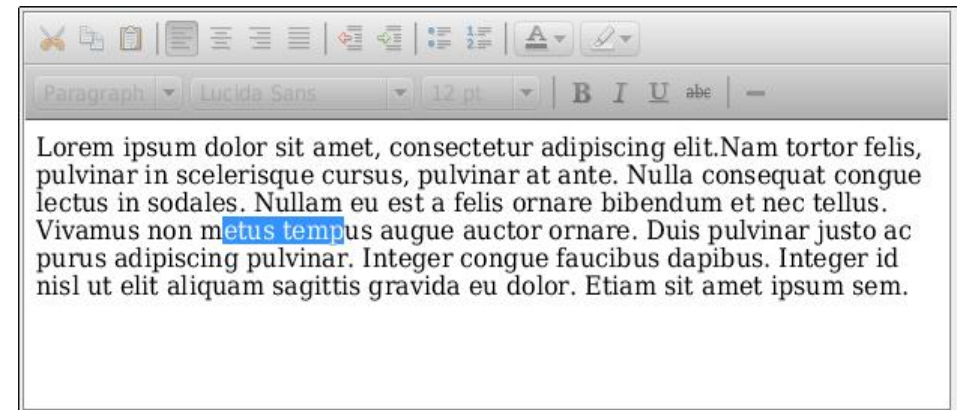
- Les divers API

- Label
- TextField
- TextFlow
- HTMLEditor



- Les possibilités

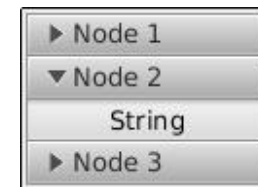
- Edition de texte pseudo-HTML
- Personnalisation par CSS
- Contrôle fin sur le rendu



LES COMPOSANTS DE JAVAFX

Barres et groupes

- MenuBar, Menu, MenuItem
- Pagination
- ToolBar (avec style CSS)
- TabPane, Tab
- Accordion
- SplitPane
 - Division cachée via css, 2 régions



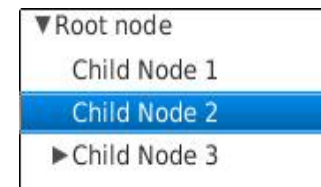
LES COMPOSANTS DE JAVAFX

Les tableaux et dérivés

- **ListView**
 - Le plus classique
 - Possibilité de personnaliser les colonnes
- **TreeView**
 - Affichage hiérarchique de données
- **TableView**
 - Le plus complet
 - Colonnes et lignes
 - Entièrement personnalisable



\$100.00
(\$12.34)
\$33.01
\$71.00
\$23,000.00
(\$6.00)
\$0.00
\$42,223.00



▼ Root node
Child Node 1
Child Node 2
▶ Child Node 3

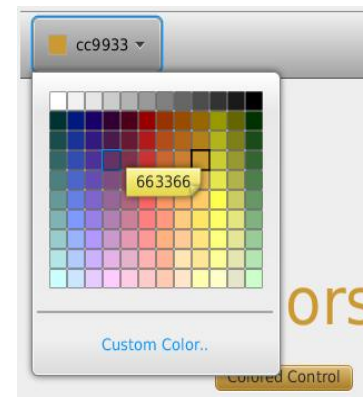
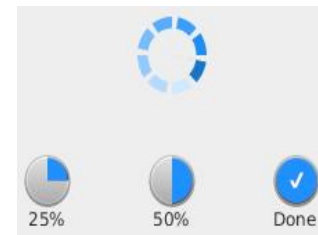
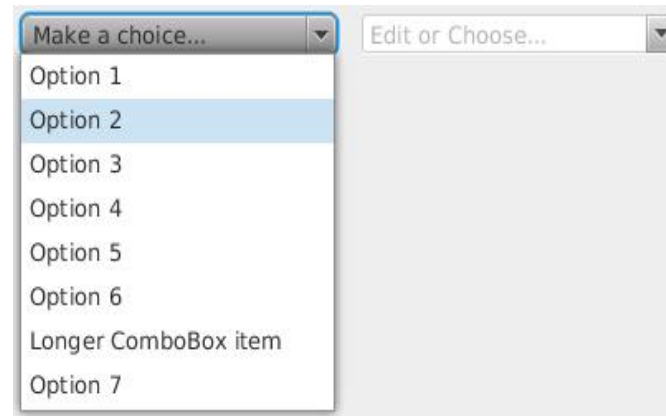


Invited	First	Last	Email
<input checked="" type="checkbox"/>	Jacob	Smith	jacob.smith@example.com
<input type="checkbox"/>	Isabella	Johnson	isabella.johnson@example.com
<input checked="" type="checkbox"/>	Ethan	Williams	ethan.williams@example.com
<input checked="" type="checkbox"/>	Emma	Jones	emma.jones@example.com
<input type="checkbox"/>	Michael	Brown	michael.brown@example.com

LES COMPOSANTS DE JAVA FX

Divers

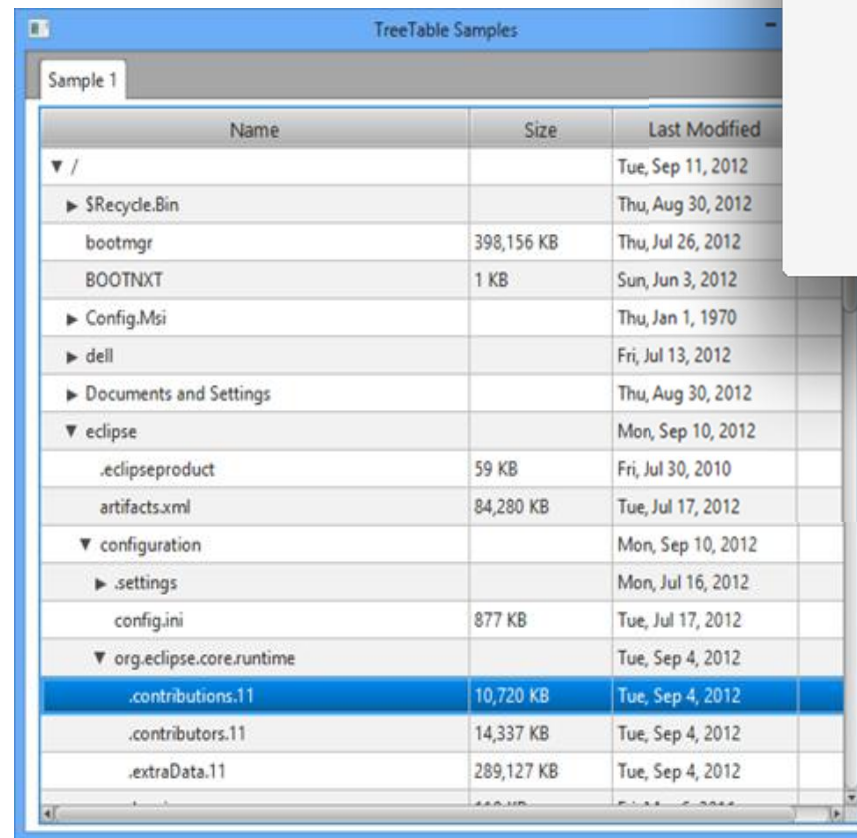
- ComboBox
- ChoiceBox
- ProgressIndicator
- ProgressBar
- ColorPicker
- DatePicker



LES COMPOSANTS DE JAVAFX

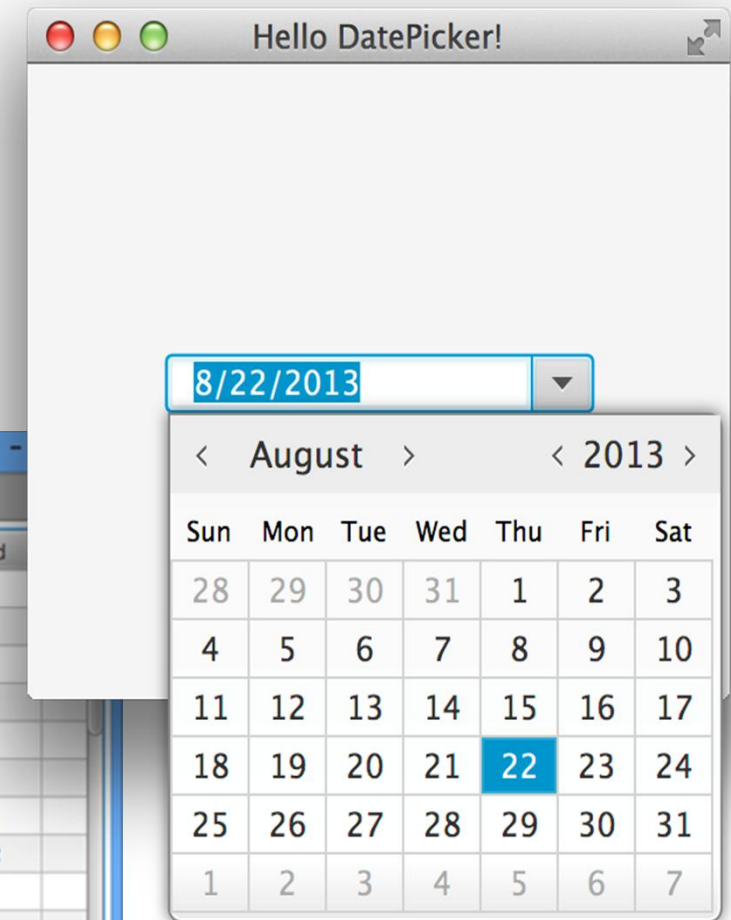
Nouveautés JavaFX 8

- DatePicker
- TreeTable



The screenshot shows a JavaFX application window titled "TreeTable Samples". It contains a TreeTable with a tab labeled "Sample 1". The table has three columns: "Name", "Size", and "Last Modified". The data is organized in a tree structure, with expandable folders indicated by arrows. The selected item is ".contributions.11".

Name	Size	Last Modified
/		Tue, Sep 11, 2012
▶ \$Recycle.Bin		Thu, Aug 30, 2012
bootmgr	398,156 KB	Thu, Jul 26, 2012
BOOTNXT	1 KB	Sun, Jun 3, 2012
▶ Config.Msi		Thu, Jan 1, 1970
▶ dell		Fri, Jul 13, 2012
▶ Documents and Settings		Thu, Aug 30, 2012
▼ eclipse		Mon, Sep 10, 2012
.eclipseproduct	59 KB	Fri, Jul 30, 2010
artifacts.xml	84,280 KB	Tue, Jul 17, 2012
▼ configuration		Mon, Sep 10, 2012
▶ .settings		Mon, Jul 16, 2012
config.ini	877 KB	Tue, Jul 17, 2012
▼ org.eclipse.core.runtime		Tue, Sep 4, 2012
.contributions.11	10,720 KB	Tue, Sep 4, 2012
.contributors.11	14,337 KB	Tue, Sep 4, 2012
.extraData.11	289,127 KB	Tue, Sep 4, 2012



LE SUPPORT MULTIMÉDIA

L'intégration facile

- Audio avec lecture MP3
 - Vidéo (FLV, Mpeg-4 avec H.264)
 - Ressource chargée dans un Media
 - Contrôle de la lecture via MediaPlayer
 - Affichage dans l'IHM via MediaView
 - (la barre de contrôle n'est pas incluse)
-
- Malheureusement: pas de support de WebCam!



LE COMPOSANT WEBVIEW

Un navigateur Web embedded!

- Rendu HTML5 complet
- Support des WebSockets
- Basé sur WebKit
- Interactions possibles entre Java et JavaScript

```
final WebEngine webEngine = webView.getEngine();  
webEngine.load("http://www.toto.co.jp/en");
```

```
webEngine.executeScript("history.back()");  
  
//équivalent à  
JSObject history = (JSObject)  
    webEngine.executeScript("history");  
history.call("back");
```

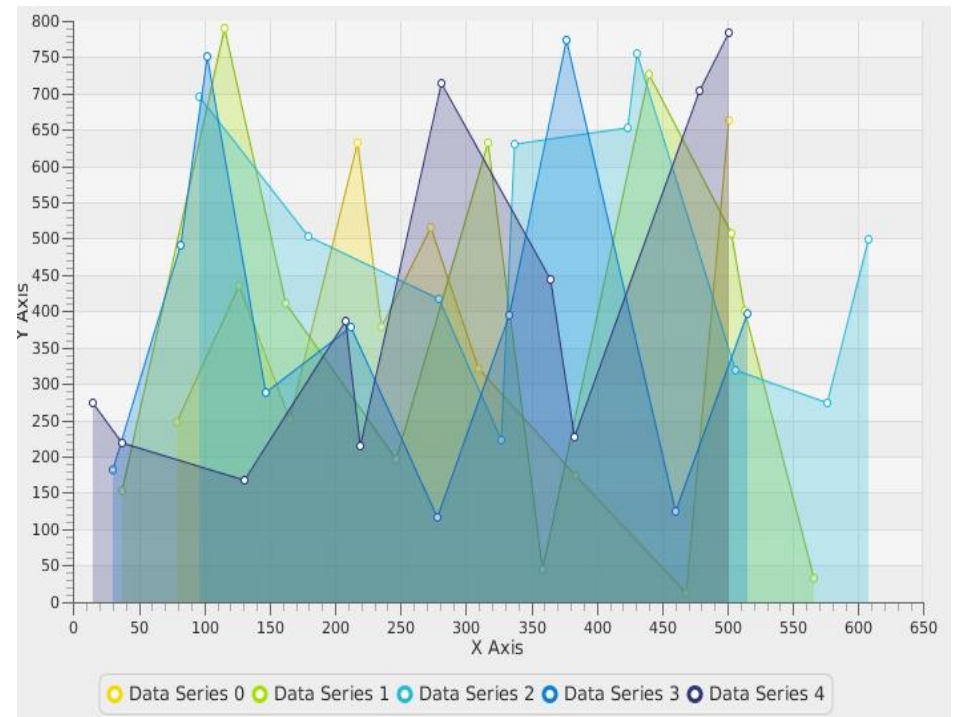


LES GRAPHES

Nombreuses possibilités

- Graphes les plus courants fournis par JavaFX
- Modèle basé sur les séries à afficher
- `XYChart.Series<TypeX, TypeY>`, `XYChart.Data<TypeX, TypeY>`

```
series.getData().add(someDataPoint);  
chart.getData().add(series);
```



ORGANISER L'AFFICHAGE

Notion de layout

- Un layout définit le positionnement et la dimension des composants
 - Il est spécifique à un conteneur
 - Il gère la répartition de l'espace restant disponible
 - Il réorganise l'affichage lors d'un redimensionnement
- Suivant les cas un layout peut
 - Gérer des données de positionnement absolues ou relatives
 - Prendre en compte l'ordre d'insertion des composants

LAYOUT PAR ANCRAGE

AnchorPane, pour les cas simples

- 4 points d'ancrage (*anchor*) sont définis : top, left, bottom, right
 - Chaque composant est attaché à une ancre et s'affiche dans la partie choisie
 - La taille préférée d'un composant est honorée
 - Un composant peut être ancré à plusieurs ancres et va alors se redimensionner pour prendre toute la largeur ou toute la longueur (left + right, top + bottom)
- Ce layout est simple mais assez limité

LAYOUT PAR RÉGION D'ÉCRAN

BorderPane, redimensionnement intelligent

- 5 régions sont définies
 - Top, prend toute la largeur en haut
 - Bottom, prend toute la largeur en bas
 - Center, zone centrale
 - Left, même hauteur que center, à gauche
 - Right, même hauteur que center, à droite
- Chaque composant est affecté à une région
 - Chaque composant est redimensionné pour occuper la région au maximum
 - Le BorderPane essaye d'honorer les tailles définies (minimum, préférée, maximum) de ses composants

LAYOUT VERTICAL ET HORIZONTAL

Classique mais efficace

- VBox et HBox
 - Les composants sont alignés horizontalement (HBox) ou verticalement (VBox)
 - L'espace est divisé en colonnes égales, dimensionnées pour accueillir le plus large / haut des composants
- FlowPane
 - Représente un conteneur orienté verticalement ou horizontalement
 - Si le nombre de composants dépasse le nombre fixé de colonnes / lignes, on passe à une nouvelle ligne / colonne

LAYOUT EN GRILLE

TilePane et GridPane

- TilePane et ses « dalles » identiques (*tiles*)
 - Peut être orienté horizontalement ou verticalement (ordre de remplissage)
 - On indique le nombre désiré de colonnes / lignes
 - Chaque dalle a la même taille
 - Chaque dalle est dimensionnée à la taille "préférée" du composant le plus volumineux
 - La grille s'ajuste au redimensionnement (ajout / suppression de lignes ou colonnes)
- GridPane et ses cellules ajustables
 - L'espace est divisible en un nombre arbitraire de colonnes et de lignes de toutes tailles
 - Possibilité de "merge" entre cellules (span)
 - Alignement dans la cellule paramétrable
 - Marge ajustable

LAYOUT PAR SUPERPOSITION

StackPane

- Les composants sont empilés les uns par dessus les autres
 - Tout les composants sont visibles mais certains masquent les autres
 - L'ordre d'insertion des composants définit leurs positions sur l'axe Z
 - Utile pour combiner plusieurs composants (texte par dessus une image par exemple)
 - Il est possible en plus de jouer sur l'opacité des éléments pour des rendus graphiques intéressants

TAILLE DES ÉLÉMENTS

Minimum, maximum, préférée

- Tous les éléments dessinés possèdent trois valeurs de dimension pour chaque coordonnée (width et height)
 - `minSize` : taille minimale du composant
 - `maxSize` : taille maximale du composant
 - `prefSize` : taille par défaut, si le composant n'a pas de dimension explicite, cette valeur sera utilisée
- Pour qu'un élément puisse utiliser tout l'espace disponible, il faut utiliser les paramètres `VGrow` et `HGrow`
 - Par défaut `maxSize = prefSize`, il faut donc modifier `maxSize` à la valeur `Infinity` pour autoriser la croissance du composant
 - Appel de la méthode statique du layout parent
Exemple : `VBox.setVgrow(node, Priority.ALWAYS);`

EXEMPLE D'UTILISATION

```
<fx:root type="javafx.scene.layout.GridPane" xmlns:fx="http://javafx.com/fxml"
    alignment="BASELINE_CENTER"
    maxHeight="Infinity" maxWidth="Infinity"
    hgap="25" vgap="25">
  <padding>
    <Insets top="25" bottom="25" left="25" right="25"/>
  </padding>
  <Button text="Bouton de test"
    maxWidth="Infinity" maxHeight="Infinity"
    GridPane.hgrow="ALWAYS" GridPane.vgrow="ALWAYS"
    GridPane.rowIndex="0" GridPane.columnIndex="0"/>
  <Button text="Say Hello"
    maxWidth="Infinity" maxHeight="Infinity"
    GridPane.hgrow="ALWAYS" GridPane.vgrow="ALWAYS"
    GridPane.rowIndex="0" GridPane.columnIndex="2"/>
  <Label text="Un texte de test tres tres tres tres tres tres tres tres tres long"
    maxWidth="Infinity" maxHeight="Infinity"
    GridPane.hgrow="ALWAYS" GridPane.vgrow="ALWAYS"
    GridPane.rowIndex="1" GridPane.columnIndex="0"/>
</fx:root>
```

DÉFINITION DES POPUPS

Toujours utile, aussi pour le multi-écrans

- Utilisation de la classe `javafx.stage.Popup`
 - `show()` permet d'ouvrir la popup
 - `hide()` permet de fermer la popup
- Utilisation de la classe `javafx.stage.Stage`
 - Possibilité d'instancier un nouveau Stage possédant sa propre Scene
 - Utile pour les applications multi-fenêtres et multi-écrans

EXEMPLE DE POPUP

```
public class PopupExample extends Application {

    @Override
    public void start(final Stage primaryStage) {

        primaryStage.setTitle("PopupExample Example");
        final Popup popupExample = new Popup();
        popupExample.setX(300);
        popupExample.setY(200);
        popupExample.getContent().addAll(new Circle(25, 25, 50, Color.AQUAMARINE));

        Button show = new Button("Show");
        show.setOnAction(actionEvent -> popupExample.show(primaryStage));

        Button hide = new Button("Hide");
        hide.setOnAction(event -> popupExample.hide());

        HBox layout = new HBox(10);
        layout.getChildren().addAll(show, hide);
        primaryStage.setScene(new Scene(layout));
        primaryStage.show();

    }
}
```