

L'ESSENTIEL DE JAVA FX – 1^{ère} partie

L'installation du plugin Java FX pour Eclipse et la création de projets Java FX.

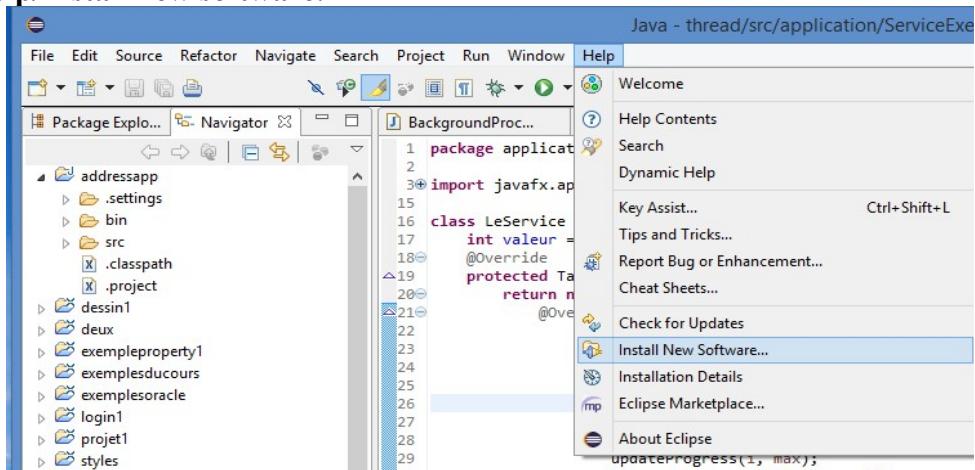
Les principes de base. Les gestionnaires de mise en page (la disposition) des composants.

Les composants de contrôle de l'interface utilisateur. La gestion des événements.

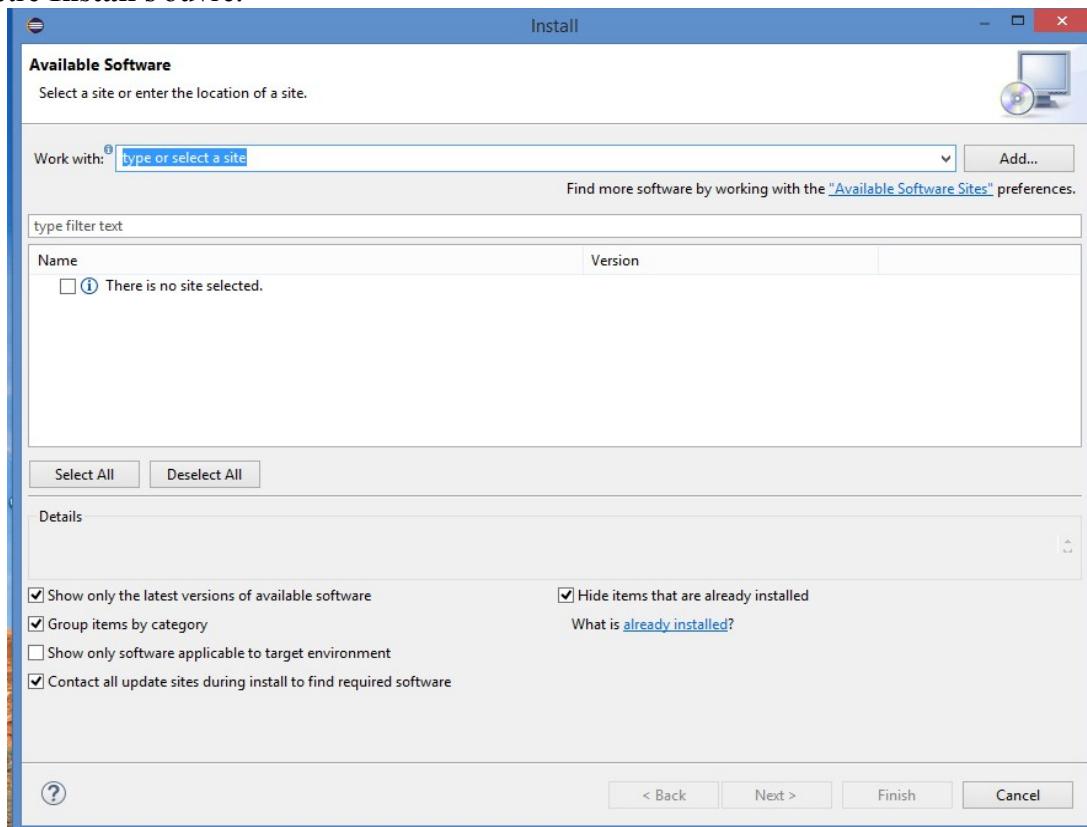
Les expressions Lambda (Java 8). Java FX et CSS.

Installation du plugin Java FX dans Eclipse

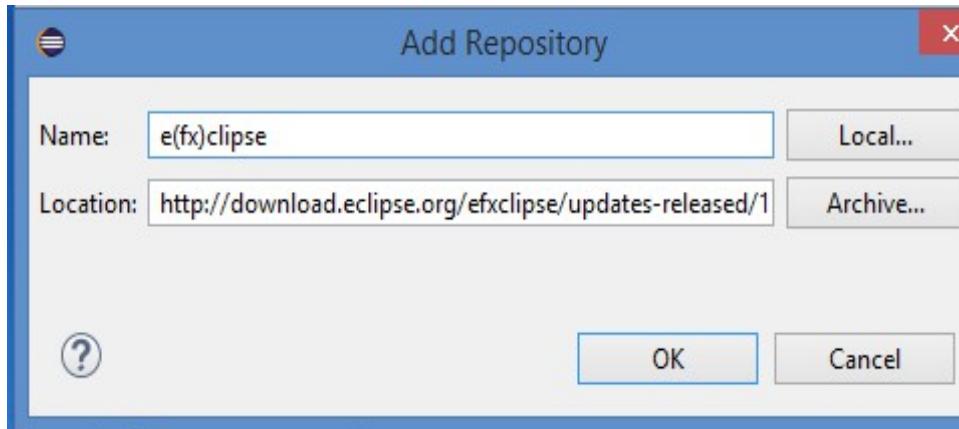
1) Faites Help/Install new software.



La fenêtre **Install** s'ouvre.

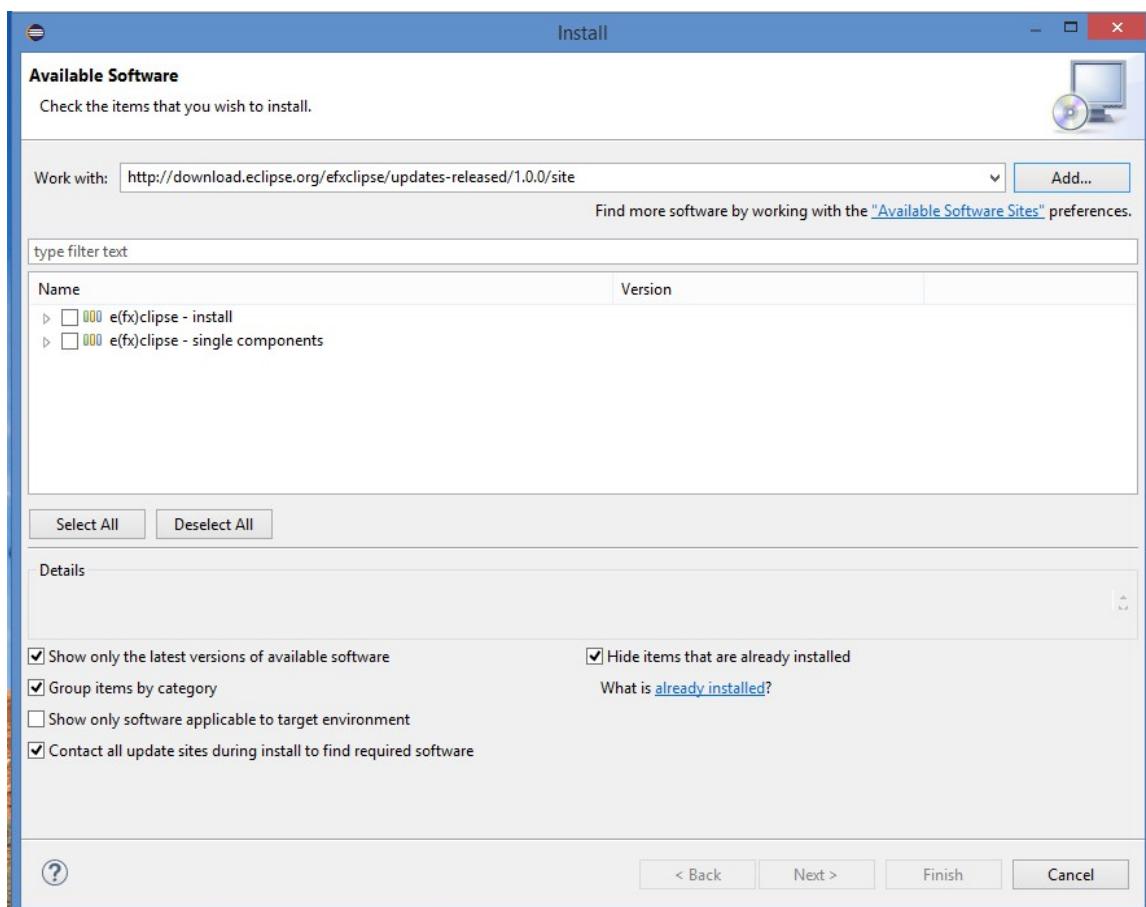


- 2) Dans le champ **Work with** tapez l'url suivante :
http://download.eclipse.org/efxclipse/updates-released/1.0.0/site
3) Puis faites **Add**. La fenêtre ci-dessous est affichée.

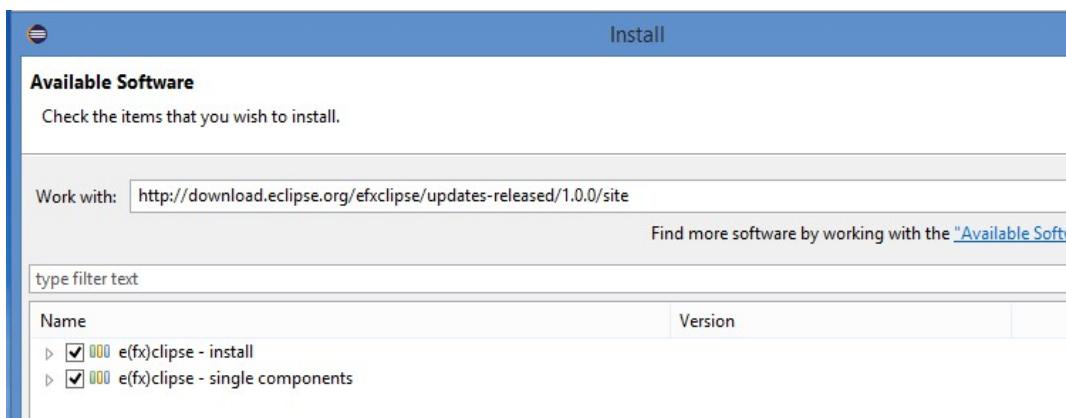


- 4) Tapez **e(fx)clipse** dans le champ **Name**, **Location** doit contenir l'url précédente.
5) Puis cliquez sur **OK**.

La fenêtre d'installation doit s'ouvrir.



- 6) Sélectionnez les 2 items en cliquant sur chaque case.



7) Puis cliquez sur **Next**.

Après un certain temps, une fenêtre affichant les détails de l'installation s'ouvre.

8) Puis cliquez sur **Next**.

9) Validez *I accept the terms of the licence agreements* dans la fenêtre affichée. Le bouton **Finish** est alors activé.

10) Puis cliquez sur **Finish**.

L'installation démarre.

Une fois l'installation terminée, Eclipse vous demande d'accepter son redémarrage pour prendre en compte le nouveau plugin.

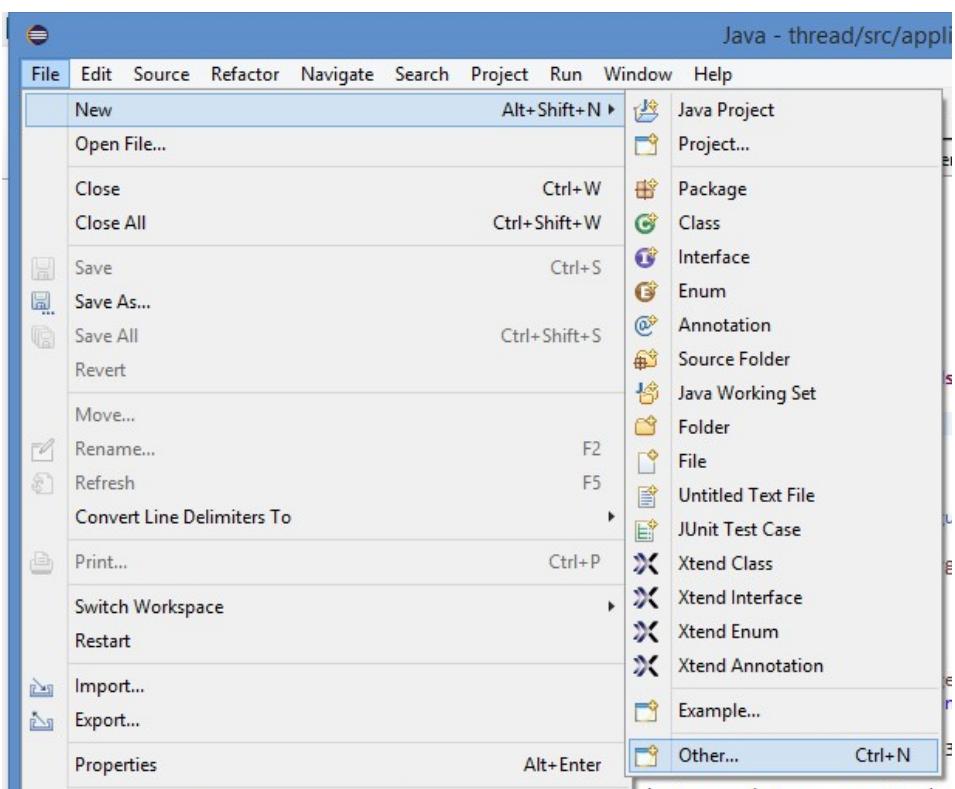
Développer en Java FX avec Eclipse

1)

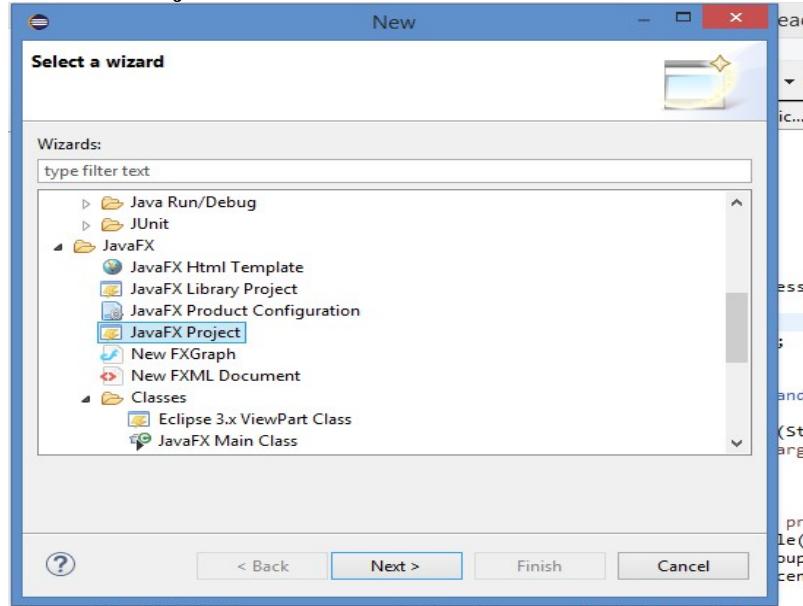
Créer un projet Java Fx.

Sélectionnez

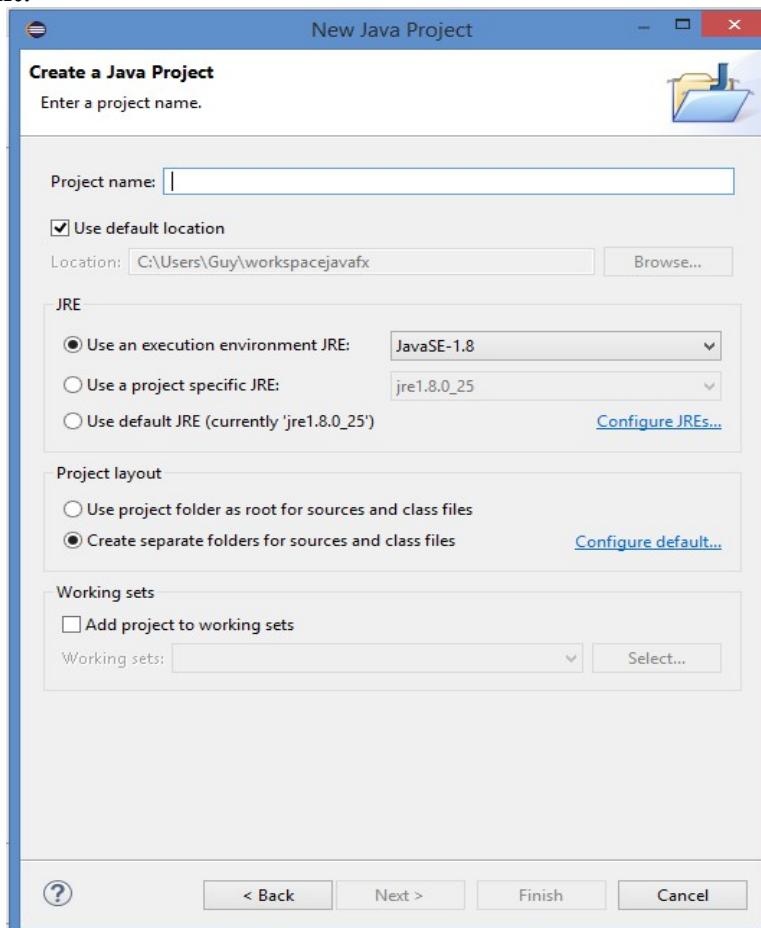
File/New/Other



Sélectionner alors **Java FX Project**



Puis cliquez sur **Next**.

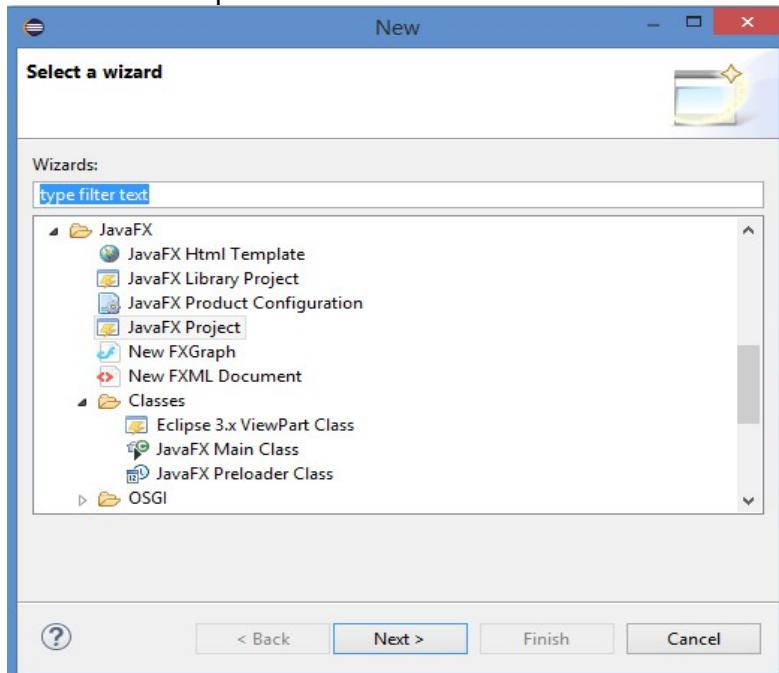


Donnez un nom au projet dans le champ **Project Name**, par exemple *premier*. Puis validez par **Finish**.

2) Créer une classe Java Fx

Sélectionnez le projet créé dans la fenêtre.

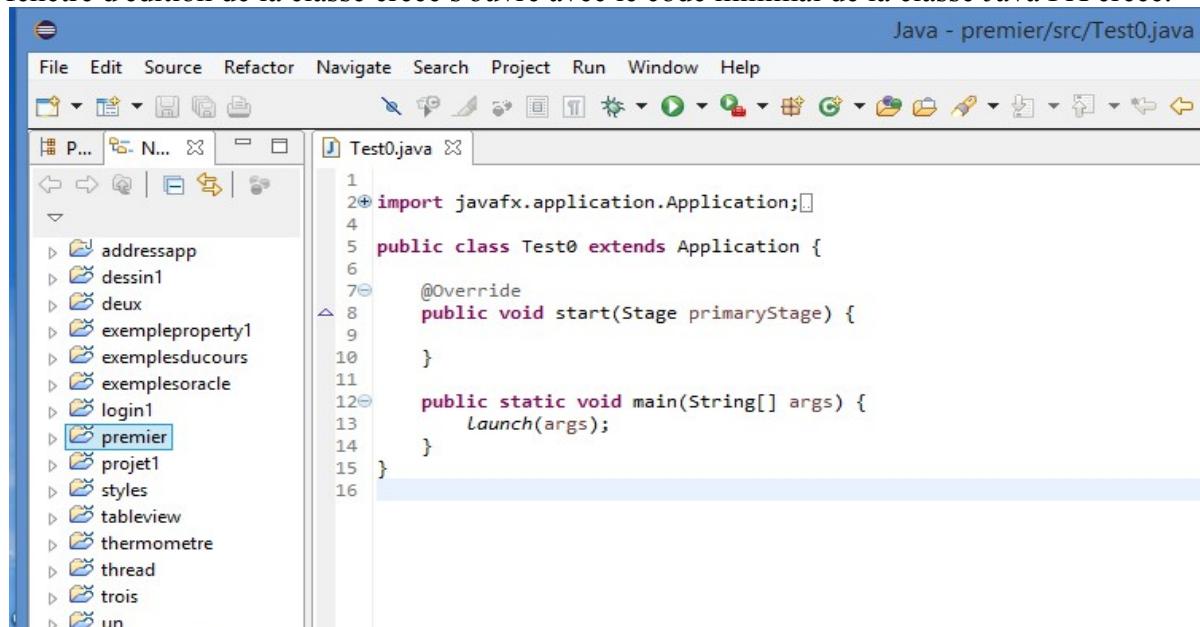
Bouton droit de la souris puis New/Other



Faites défiler la fenêtre et sélectionner **Java FX Main class** puis **Next**.

Donnez un nom à la classe, par exemple *Test0* puis cliquez sur **Finish**.

La fenêtre d'édition de la classe créée s'ouvre avec le code minimal de la classe Java FX créée.



Les principes de base de Java FX

Java FX est la dernière API java développée par Oracle pour réaliser les applications graphiques appelées interfaces utilisateurs (UI User Interface) ou IHM. Le développement de Java FX a débuté en 2008 et nous sommes actuellement à la version Java FX 8.

Java FX propose dans une même API le développement d'applications riches: interface utilisateur, gestion des médias, 2D, 3D, animation 3D, application Web.

Java FX introduit également de nouveaux concepts comme les Properties et le Binding.

Les expressions lambda sont elles un nouveau concept de Java 8.

La fenêtre Java FX

Le concept repose sur celui d'une *fenêtre théâtre* – objet de type **Stage** – contenant une *scène* – objet de type **Scene** – . La *scène* contient un *décors* constitué de tous les composants à afficher.

La fenêtre principale de l'UI est assimilable au *théâtre* – objet de type **Stage** –

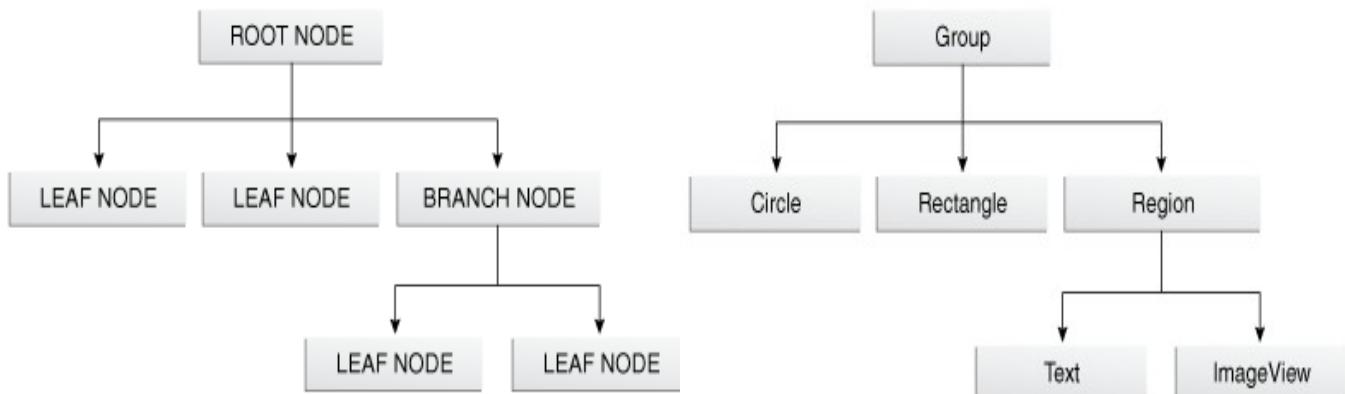
Organisation de la scène

La scène contient un décors.

Le décors est construit à partir d'un **nœud racine**.

Le **nœud racine** est le plus souvent:

- soit un **gestionnaire de disposition** dans le cas où l'interface contient des contrôles classiques comme des boutons, des cases à cocher, des labels...
- soit un objet de type **Group** dans le cas d'une application graphique.



Java FX définit quelles classes peuvent être **nœud racine** et quelles autres peuvent être **nœud branche ou feuille**.

Le décors de la figure de droite ci-dessus contient un nœud racine de type **Group**, lui-même contenant 2 objets nœuds feuilles **Circle** et **Rectangle** et un autre objet nœud branche de type **Region**.

On peut considérer le nœud racine comme le décors de la scène.

Tous les composants proposés par Java FX pour construire le décors sont des objets instances de classes dérivées de la classe **Node**. A ce titre, ils peuvent tous subir les transformations suivantes : translation, rotation autour d'un pivot, rotation autour d'un axe, changement d'échelle.

1^{er} exemple d'un dessin constitué de 4 lignes et d'un rectangle :

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeLineCap;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class DrawLines extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Titre de la fenêtre
        primaryStage.setTitle("Dessins de 4 lignes et d'un rectangle");

        // Création du décors
        Group root = new Group();
        // On place le décors dans la scène + taille et couleur de fond de la scène
        Scene scene = new Scene(root,500,300,Color.GRAY);

        // On remplit le décors...
        Line ligne1 = new Line(10,10,200,50);
        // avec une 1ère ligne
        root.getChildren().add(ligne1);

        Line ligne2 = new Line(10,50,200,110);
        ligne2.setStroke(Color.PINK); // couleur rose
        ligne2.setStrokeWidth(10); // largeur du trait
        ligne2.setStrokeLineCap(StrokeLineCap.ROUND); // extrémités arrondies
        // avec une 2ème ligne
        root.getChildren().add(ligne2);

        Text texte = new Text(20,180,"Une ligne bleue");
        root.getChildren().add(texte);
        Line ligne3 = new Line(20,250,220,150);
        ligne3.setStroke(Color.BLUE);
        ligne3.setStrokeWidth(15);
        ligne3.setStrokeLineCap(StrokeLineCap.SQUARE); // extrémités carrées
        // avec une 3ème ligne
        root.getChildren().add(ligne3);

        Line ligne4 = new Line(250,250,450,280);
        ligne4.setStroke(Color.CORAL); // couleur corail
        ligne4.setStrokeWidth(5); // largeur du trait
        ligne4.setStrokeLineCap(StrokeLineCap.BUTT); // extrémités carrées
        ligne4.setStrokeDashOffset(10);
        ligne4.getStrokeDashArray().addAll(15d,5d); // traits tiretés
```

```

// avec une 4ème ligne
root.getChildren().add(ligne4);

Rectangle rect = new Rectangle(300,10,80,40);
rect.setFill(Color.RED);
// avec un rectangle
root.getChildren().add(rect);

// On place la scène dans la fenêtre théâtre
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
}
}

```

Quelques explications importantes :

// Une classe Java Fx hérite obligatoirement de la classe Application
public class DrawLines extends Application {

// La méthode **public void start(Stage primaryStage)** doit être redéfinie dans toute classe
// d'application Java FX. Elle constitue le point de départ de l'exécution de notre application.
// Elle reçoit en argument l'objet de type **Stage** (la fenêtre théâtre **primaryStage**) à utiliser dans
// notre application.

// Crédit du décors en commençant par le nœud racine :

Group root = new Group();

// On place le décors dans la scène

Scene scene = new Scene(root,500,300,Color.GRAY);

// On remplit le décors

Line ligne1 = new Line(10,10,200,50);

root.getChildren().add(ligne1);

...

root.getChildren().add(...);

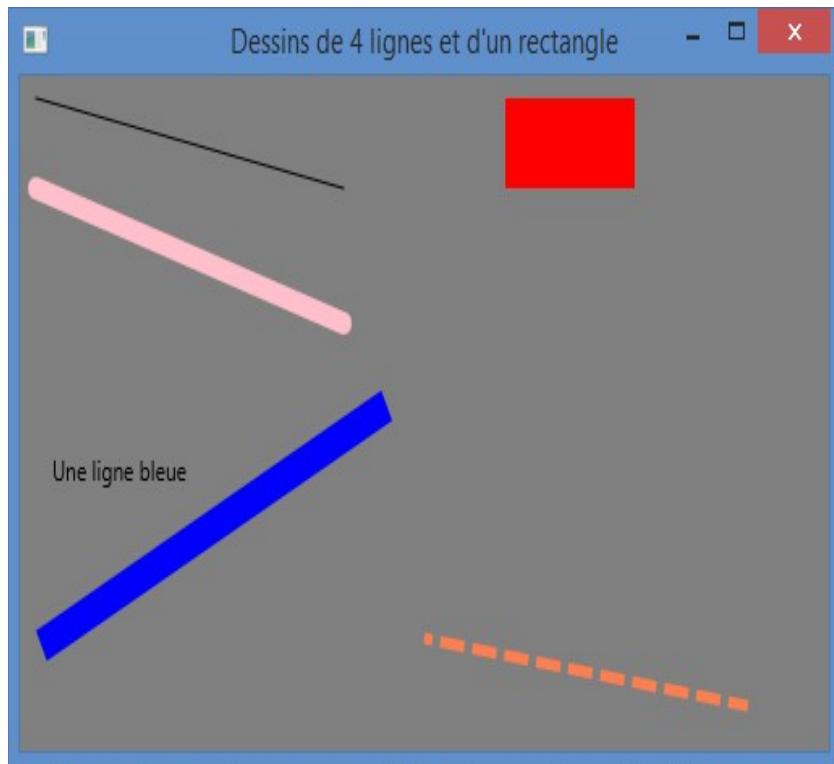
// On place la scène dans la fenêtre théâtre

primaryStage.setScene(scene);

// On affiche la fenêtre

primaryStage.show();

// La méthode **main** est nécessaire pour exécuter l'application en ligne de commande.



Les gestionnaires de disposition - de mise en page

Les gestionnaires de mise en page – **Layout Manager** – permettent de choisir l'emplacement des nœuds – souvent des composants (bouton, label, champ de saisie...) – dans une fenêtre.

BorderPane

La disposition est du type «géométrique». Un BorderPane contient 5 régions où placer les nœuds. Cette disposition est très courante et commune aux interfaces rencontrées sur Internet. Le menu ou les liens de navigation peuvent être placés en haut ou à droite ou à gauche, la région centrale contenant la vue de la sélection. Le bas contenant des images avec des liens supplémentaires.

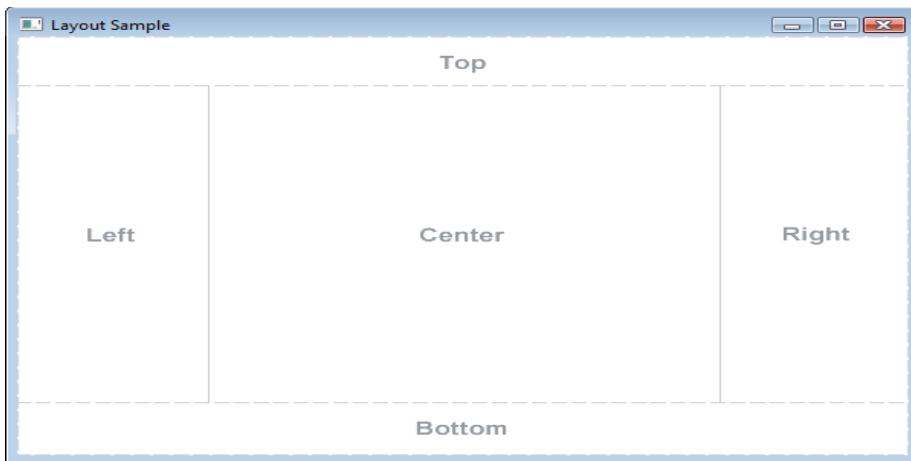
La classe **BorderPane** contient 5 méthodes utilisées pour positionner les composants :

- **setTop(node)**,
- **setRight(node)**,
- **setLeft(node)**,
- **setCenter(node)**
- **setBottom(node)**.

La méthode **setAlignment()** permet de choisir l'alignement du nœud dans la zone concernée, la méthode **setMargin()** la marge autour du nœud considéré.

Exemple :

```
ListView list = new ListView() ;  
BorderPane.setAlignment(list, Pos.TOP_LEFT) ;  
BorderPane.setMargin(list, new Insets(12,12,12,12));  
borderPane.setCenter(list);
```



2^{ème} exemple : création d'une fenêtre vide pour y ajouter des objets contrôles

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class BorderPaneEx1 extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Titre de la fenêtre
        primaryStage.setTitle("Un exemple de BorderPane");
        // Création de la racine - décors -
        BorderPane root = new BorderPane();
        // Création de la scène, association du décors et de la scène, taille de la scène
        Scene scene = new Scene(root,400,300);
        // On place la scène dans la fenêtre
        primaryStage.setScene(scene);
        // On affiche la fenêtre
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



3^{ème} exemple : ajout d'un bouton au centre du décors de la fenêtre du 2^{ème} exemple

Les instructions suivantes écrites en gras et insérées comme indiqué permettent d'ajouter un bouton au centre du décors.

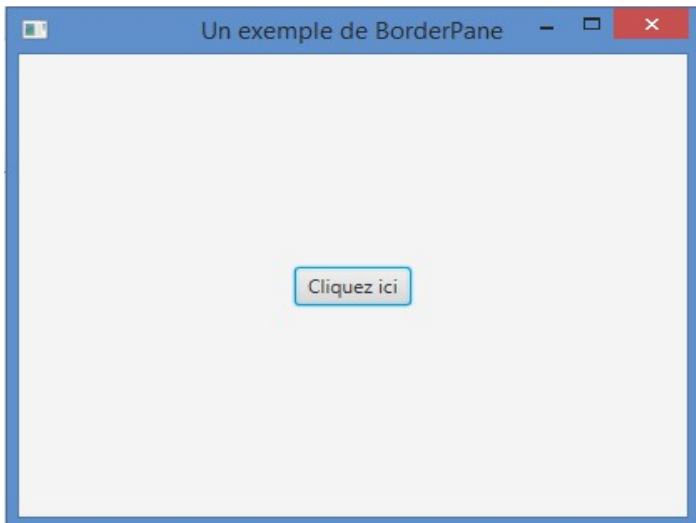
```

// Cr ation de la racine - decors -
BorderPane root = new BorderPane();

// Insertion des 2 instructions suivantes pour ajouter le bouton au centre du d cors
Button bouton = new Button("Cliquez ici");
root.setCenter(bouton);

// Cr ation de la sc ne, association du d cors et de la sc ne, taille de la sc ne
Scene scene = new Scene(root,400,300);

```



Hbox

Les noeuds contenus dans un gestionnaire **Hbox** sont dispos s horizontalement.

4 me exemple

```

package application;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class HboxExemple extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("HBox avec rectangles");
        //HBox avec un espace de 10 pixels entre les composants contenus
        HBox hbox = new HBox(10);

        //Marge de 5 pixels entre la HBox et la bordure de la fen tre
        hbox.setPadding(new Insets(5));
    }
}

```

```

        Rectangle r1 = new Rectangle(50,50) ;
        Rectangle r2 = new Rectangle(100,100) ;
        Rectangle r3 = new Rectangle(25,100) ;
        Rectangle r4 = new Rectangle(100,25) ;

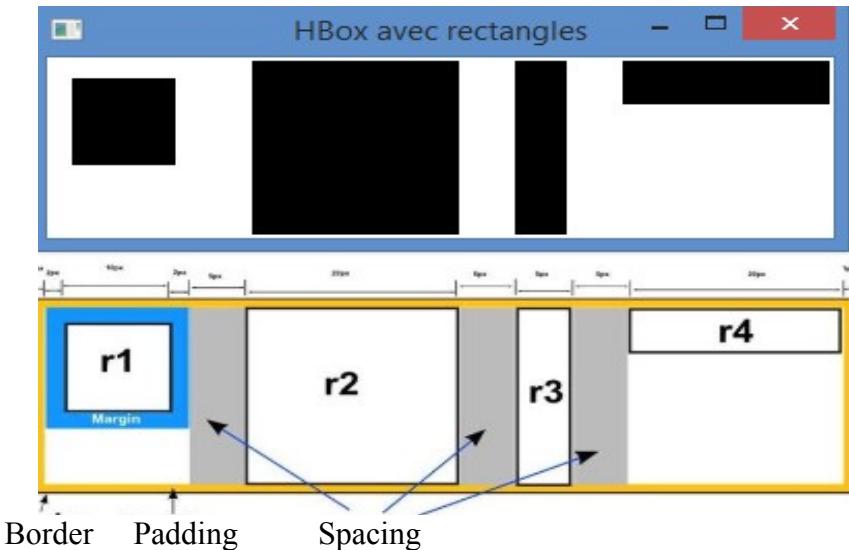
        // Marge supplémentaire de 10 pixels entre le 1er rectangle et les autres éléments
        HBox.setMargin(r1, new Insets(10,10,10,10));

        hbox.getChildren().addAll(r1,r2,r3,r4);

        Scene scene = new Scene(hbox);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```



Vbox

Les noeuds contenus dans un gestionnaire **Vbox** sont disposés verticalement.

Le fragment de code ci-dessous est similaire à l'exemple précédent mais dans une Vbox

5^{ème} exemple

```

public void start(Stage primaryStage) {
    primaryStage.setTitle("VBox avec rectangles");
    //VBox avec un espace de 27 pixels entre les composants
    VBox vbox = new VBox(27) ;

    //Marge de 2 pixels entre la VBox et la bordure de la fenêtre
}

```

```

vbox.setPadding(new Insets(2));

Rectangle r1 = new Rectangle(50,50) ;
Rectangle r2 = new Rectangle(100,100) ;
Rectangle r3 = new Rectangle(25,100) ;
Rectangle r4 = new Rectangle(100,25) ;

// Marge supplémentaire de 10 pixels entre le 1er rectangle et les autres éléments
vbox.setMargin(r1, new Insets(10,10,10,10));

vbox.getChildren().addAll(r1,r2,r3,r4);

Scene scene = new Scene(vbox);
primaryStage.setScene(scene);
primaryStage.show();
}

```

FlowPane

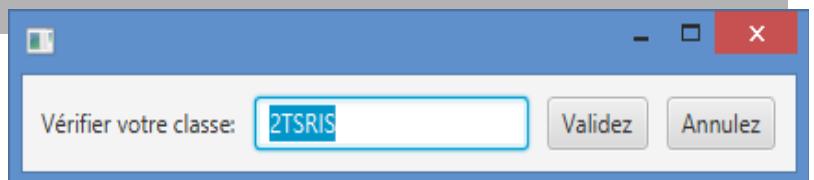
Le **FlowPane** dispose par défaut les nœuds de gauche à droite en ligne jusqu'au bord droit de la fenêtre puis les composants suivants sur une 2^{ème} ligne.

6^{ème} exemple

```
public class FlowPaneExemple extends Application {
```

```

@Override
public void start(Stage primaryStage) {
    //Espace horizontal et vertical de 10 entre les composants
    FlowPane flow = new FlowPane(10,10) ;
    //Marge de 10 entre le FlowPane et le cadre de la fenêtre
    flow.setPadding(new Insets(10));
    Label label = new Label("Vérifier votre classe:");
    TextField nom = new TextField("2TSIRIS");
    Button b1 = new Button("Validez");
    Button b2 = new Button("Annulez");
    flow.getChildren().addAll(label,nom,b1,b2);
    Scene scene = new Scene(flow);
    primaryStage.setScene(scene);
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```



On peut créer un FlowPane vertical :

```
FlowPane flow = new FlowPane(Orientation.VERTICAL);
```

GridPane

Les nœuds sont disposés dans une grille flexible. On ne précise ni le nombre de lignes, ni le nombre de colonnes, ni le nombre de cellules à la création de la grille, elle s'adaptera automatiquement lors du placement des composants.

Chaque composant est disposé dans une cellule de la grille dont il faut préciser la colonne et la ligne. On peut également préciser l'étalement (span) éventuel du composant sur plusieurs cellules de la grille.

7^{ème} exemple

```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Bienvenue extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX Bienvenue");

        GridPane grid = new GridPane();
        // La grille est centrée dans le panneau
        grid.setAlignment(Pos.CENTER);
        // Largeur de l'espacement horizontal entre les cellules
        grid.setHgap(10);
        // Largeur de l'espacement vertical entre les cellules
        grid.setVgap(10);
        // Marge entre la grille et la bordure de la fenêtre
        grid.setPadding(new Insets(25, 25, 25, 25));

        Text scenetitle = new Text("Bienvenue sur votre site préféré");
        scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
        // Ajout du texte scenetitle en colonne 0 et ligne 0 étalé sur 2 colonnes et 1 ligne
        grid.add(scenetitle, 0, 0, 2, 1);

        Label userName = new Label("Nom d'utilisateur:");
        // Ajout du label userName en colonne 0 et ligne 1
        grid.add(userName, 0, 1);
```

```

        TextField userTextField = new TextField();
        //Ajout du champ userTextField en colonne 1 et ligne 1
        grid.add(userTextField, 1, 1);

        Label pw = new Label("Mot de passe:");
        //Ajout du label pw en colonne 0 et ligne 2
        grid.add(pw, 0, 2);

        PasswordField pwBox = new PasswordField();
        //Ajout du champ pwBox en colonne 1 et ligne 2
        grid.add(pwBox, 1, 2);

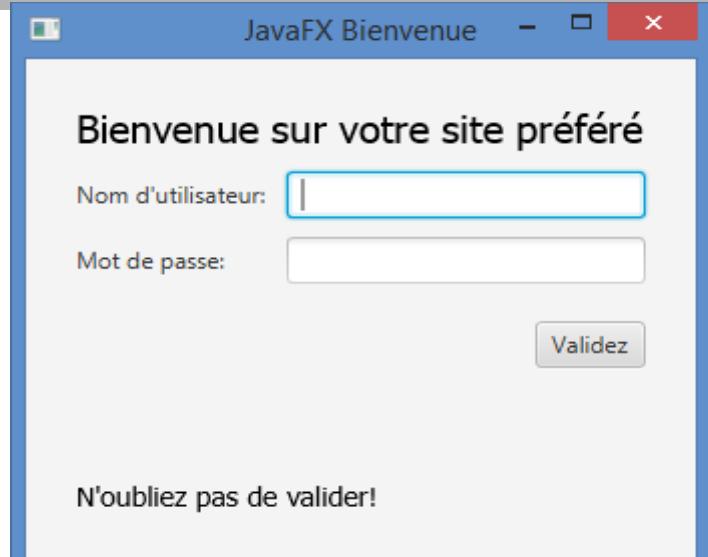
        Button btn = new Button("Validez");
        //Ajout du bouton btn en colonne 1 et ligne 4 et justifié à droite
        GridPane.setAlignment(btn, HPos.RIGHT) ;
        grid.add(btn, 1, 4);

        Text message = new Text("N'oubliez pas de valider!");
        message.setFont(Font.font("Tahoma", FontWeight.NORMAL, 14));
        //Ajout du texte message en colonne 0 et ligne 10 étalé sur 2 colonnes et 1 ligne
        grid.add(message, 0, 10, 2, 1);

        Scene scene = new Scene(grid, 320, 275);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```



AnchorPane

Un **AnchorPane** permet d'ancrer les bords des nœuds aux bord du AnchorPane. Il peut y avoir un ou plusieurs bords d'ancrage ainsi qu'un offset entre le bord du AnchorPane et le bord ancré du nœud. Si la fenêtre est redimensionnée, le composant maintient sa position relative par-rapport à ses voisins.

points d'ancrage.

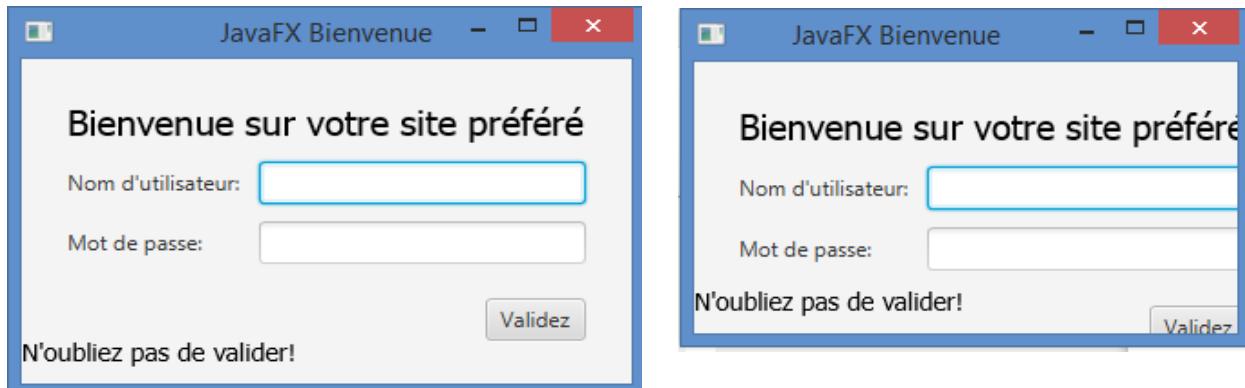
On modifie le 7^{ème} exemple, on crée un AnchorPane dans lequel on place séparément le GridPane et le dernier message "N'oubliez pas de valider!".

8^{ème} exemple

```
public void start(Stage primaryStage) {  
    primaryStage.setTitle("JavaFX Bienvenue");  
    GridPane grid = new GridPane();  
    grid.setAlignment(Pos.CENTER);  
    grid.setHgap(10);  
    grid.setVgap(10);  
    grid.setPadding(new Insets(25, 25, 25, 25));  
  
    Text scenetitle = new Text("Bienvenue sur votre site préféré");  
    scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));  
    grid.add(scenetitle, 0, 0, 2, 1);  
  
    Label userName = new Label("Nom d'utilisateur");  
    grid.add(userName, 0, 1);  
  
    TextField userTextField = new TextField();  
    grid.add(userTextField, 1, 1);  
  
    Label pw = new Label("Mot de passe");  
    grid.add(pw, 0, 2);  
  
    PasswordField pwBox = new PasswordField();  
    grid.add(pwBox, 1, 2);  
  
    Button btn = new Button("Validez");  
    GridPane.setHalignment(btn, HPos.RIGHT);  
    grid.add(btn, 1, 4);  
  
    Text message = new Text("N'oubliez pas de valider!");  
    message.setFont(Font.font("Tahoma", FontWeight.NORMAL, 14));  
  
    // On crée le AnchorPane  
    AnchorPane ancre = new AnchorPane();  
    // On ajoute séparément la grille et le message  
    ancre.getChildren().addAll(grid, message);  
    // On ancre la grille en haut  
    AnchorPane.setTopAnchor(grid, 0.0);  
    // On ancre le message en bas  
    AnchorPane.setBottomAnchor(message, 10.0);  
  
    Scene scene = new Scene(ancre);  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```

Réduction de la taille de la fenêtre par le bas: le message remonte avec le bord bas de la fenêtre

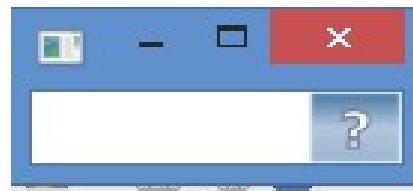
Affichage de la fenêtre



StackPane

Le StackPane place tous les nœuds dans une pile unique avec chaque nœud mis au-dessus du précédent. Ce modèle fournit un moyen facile du superposer du texte sur une forme ou une image ou faire chevaucher des formes communes pour créer une forme complexe.

L'exemple suivant tiré de la documentation Oracle affiche une icône d'aide créée en empilant un point d'interrogation sur un rectangle rempli avec un fond dégradé.



9ème exemple

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class StackPaneExemple extends Application {

    @Override
    public void start(Stage primaryStage) {
        StackPane stack = new StackPane();
        Rectangle helpIcon = new Rectangle(30.0, 25.0);
        helpIcon.setFill(new LinearGradient(0,0,0,1, true, CycleMethod.NO_CYCLE,
            new Stop[]{


```

```

        new Stop(0,Color.web("#4977A3")),
        new Stop(0.5, Color.web("#B0C6DA")),
        new Stop(1,Color.web("#9CB6CF")),}));}

Text helpText = new Text("?");
helpText.setFont(Font.font("Verdana", FontWeight.BOLD, 18));
//Définit la couleur de l'intérieur de la forme
helpText.setFill(Color.WHITE);
//Définit la couleur du trait qui entoure (marque) la forme
helpText.setStroke(Color.web("#7080A0"));

stack.getChildren().addAll(helpIcon, helpText);
stack.setAlignment(Pos.CENTER_RIGHT); // Right-justify nodes in stack
StackPane.setMargin(helpText, new Insets(0, 10, 0, 0)); // Center "?"

Scene scene = new Scene(stack);
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
}
}

```

TilePane

Le fonctionnement d'un **TilePane** est similaire au **FlowPane**. Le **TilePane** place les nœuds dans une grille où chaque cellule a la même taille. Les nœuds peuvent être disposés horizontalement ou verticalement.

Les composants de contrôle de l'interface utilisateur

Java FX propose tous les contrôles habituels: Label, Bouton, Bouton Radio, Case à cocher, Liste déroulante, Champ de saisie, Menu...

Tous les contrôles de l'interface utilisateur résident dans le package **javafx.scene.control**.

Il est impossible ici de détailler tous les contrôles proposés, voir le lien

<http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/>

pour des explications détaillées et des exemples.

Tous les contrôles de l'UI héritent de la classe **Node**. Ils peuvent donc tous subir des transformations comme la rotation, la translation et le changement d'échelle.



Les labels

10^{ème} exemple

```

import javafx.scene.Group;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class LabelSample extends Application {
    Label label3 = new Label("Un label qui nécessite plusieurs lignes");

    public static void main(String[] args) {
        launch(args);
    }
}

```

```

@Override
public void start(Stage stage) {
    Scene scene = new Scene(new Group());
    stage.setTitle("Que de labels!");
    stage.setWidth(520);
    stage.setHeight(180);
    HBox hbox = new HBox();
    Image image = new Image(getClass().getResourceAsStream("1.jpg"));
    Label label1 = new Label("Recherche");
    label1.setGraphic(new ImageView(image));
    label1.setFont(new Font("Arial", 30));
    label1.setTextFill(Color.web("#0076a3"));
    label1.setTextAlignment(TextAlignment.JUSTIFY);
    Label label2 = new Label ("Valeurs");
    label2.setFont(Font.font("Cambria", 32));
    label2.setRotate(270);
    label2.setTranslateY(50);
    label3.setWrapText(true);
    label3.setTranslateY(50);
    label3.setPrefWidth(100);

    hbox.setSpacing(10);
    hbox.getChildren().add(label1);
    hbox.getChildren().add(label2);
    hbox.getChildren().add(label3);
    ((Group)scene.getRoot()).getChildren().add(hbox);
    stage.setScene(scene);
    stage.show();
}
}

```



Les menus

Tous est prévu pour créer barre de menus, menus principaux, sous-menus, sous-menus cochés, sous-menus radio, séparateur de menus.

L'exemple suivant présente la création d'une barre de menus avec 3 menus principaux. On détaille la création d'un sous-menu avec, l'un associé avec une image, l'autre avec une combinaison de touches.

11^{ème} exemple

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyCombination;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MenuExemple1 extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Exemple de menu");
        BorderPane root = new BorderPane();
        //Création de la barre de menus
        MenuBar menubar = newMenuBar();
        //Création du menu principal Fichiers
        Menu menufichier= new Menu("Fichiers");
        //Création du sous choix Nouveau associé avec une image
        MenuItem nouveau=new MenuItem("Nouveau", new ImageView("newdoc.png"));
        //Création du sous choix Ouvrir associé avec une combinaison de touches
        MenuItem ouvrir = new MenuItem("Ouvrir");
        ouvrir.setAccelerator(KeyCombination.keyCombination("Ctrl+O"));
        //Création du sous choix Enregistrer
        MenuItem sauver = new MenuItem("Enregistrer");
        //Création du sous choix Quitter
        MenuItem quitter = new MenuItem("Quitter");
        //Ajout des sous sous choix au menu Fichier avec un séparateur
        menufichier.getItems().addAll(nouveau, ouvrir, sauver,
            new SeparatorMenuItem(), quitter);

        //Création des menus principaux Vue et Alarme
        Menu menuvue = new Menu("Vue");
        Menu menualarme = new Menu("Alarme");

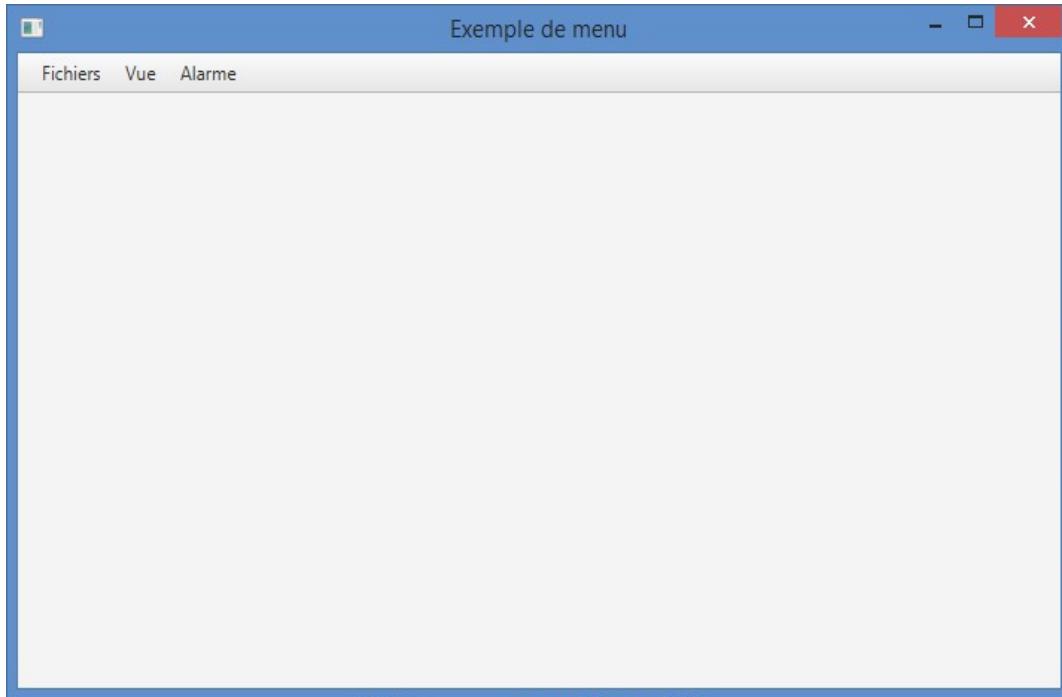
        //Ajout des menus à la barre de menus
        menubar.getMenus().addAll(menufichier, menuvue, menualarme);

        root.setTop(menubar);

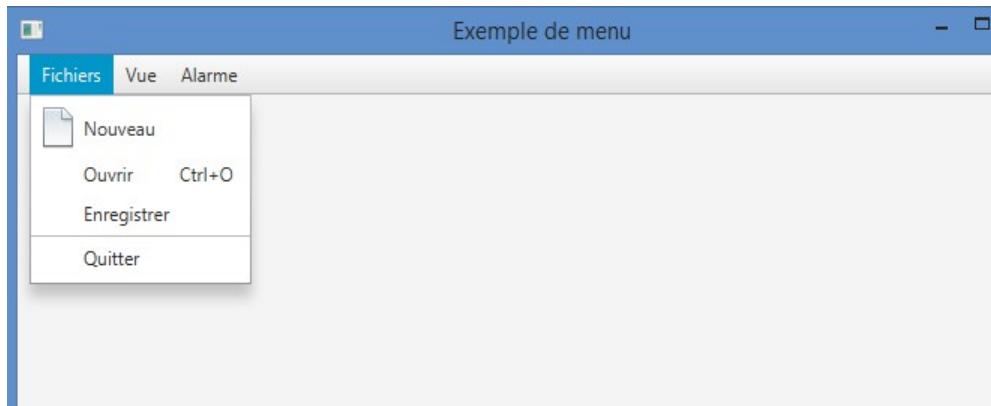
        Scene scene = new Scene(root, 700,400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
```

```
        launch(args);  
    }  
}
```



On sélectionne le menu Fichiers :



La solution suivante pour créer le raccourci est indépendante de la plateforme d'exécution.

KeyCombination.SHORTCUT_DOWN correspond à la touche Ctrl sous Windows et à la touche Meta sous MacOs

```
MenuItem ouvrir = new MenuItem("Ouvrir");  
ouvrir.setAccelerator(new KeyCodeCombination(KeyCode.O,  
KeyCombination.SHORTCUT_DOWN));
```

KeyCombination.CONTROL_DOWN correspond à la touche Ctrl sous Windows.

KeyCombination.META_DOWN correspond à la touche Ctrl sous MacOs.

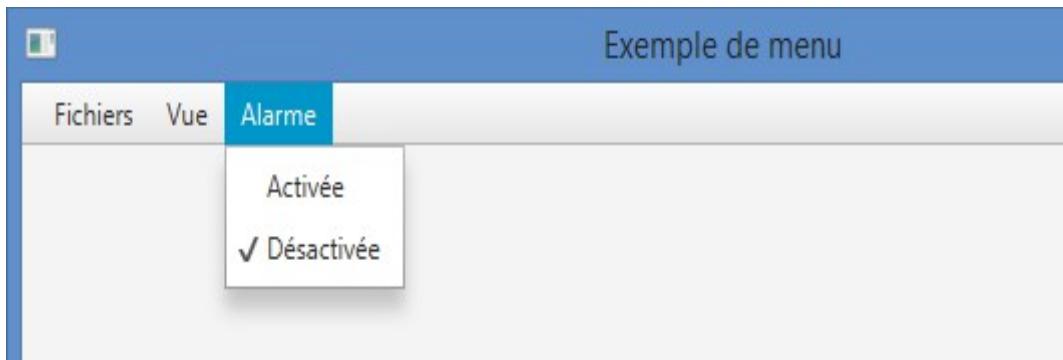
Ajout de sous-menus de type radio

12^{ème} exemple Il faut insérer les instructions écrites en gras après la création du menu menualarme.

```

Menu menualarme = new Menu("Alarme");
//Création du toggle group
ToggleGroup etat = new ToggleGroup() ;
//Création d'un sous-menu radio
RadioMenuItem active = new RadioMenuItem("Activée");
//Création d'un sous-menu radio
RadioMenuItem desactive = new RadioMenuItem("Désactivée");
//Ajout des sous-menu radio dans le toggle group
active.setToggleGroup(etat);
desactive.setToggleGroup(etat);
//Marquage par défaut du sous-menu radio desactive
desactive.setSelected(true);
//Ajout des sous-menus radio au menu alarme
menualarme.getItems().addAll(active,desactive);

```



Ajout de sous-menus de type coché

13^{ème} exemple

Il faut insérer les instructions écrites en gras après la création du menu menuvue

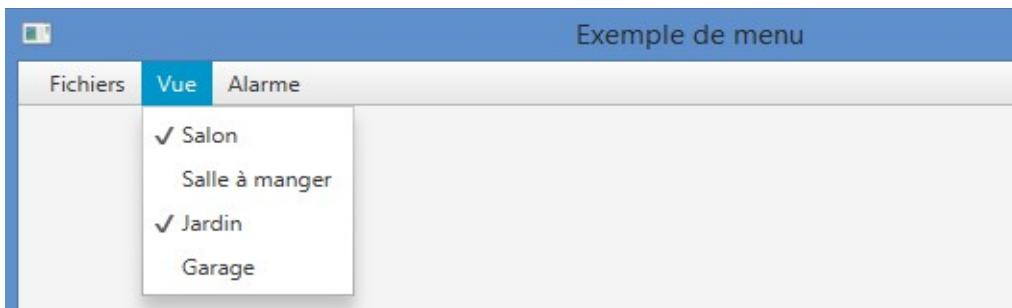
```

Menu menuvue = new Menu("Vue");
CheckMenuItem vue1 = new CheckMenuItem("Salon");
CheckMenuItem vue2 = new CheckMenuItem("Salle à manger");
CheckMenuItem vue3 = new CheckMenuItem("Jardin");
CheckMenuItem vue4 = new CheckMenuItem("Garage");

vue1.setSelected(true);
vue3.setSelected(true);

menuvue.getItems().addAll(vue1, vue2, vue3, vue4);

```



Les ListView

Une **ListView** affiche une liste verticale ou horizontale d'éléments que l'utilisateur peut sélectionner et/ou avec lesquels l'utilisateur peut inter agir.

Les éléments affichés d'une **ListView** sont obligatoirement des éléments d'une **ObservableList**. Cette **ObservableList** est automatiquement observée par la **ListView**. Ainsi, tout changement dans la **ObservableList** (par exemple l'ajout ou la suppression d'un élément) est automatiquement reporté et montré dans la **ListView**.

14^{ème} exemple

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.StackPane;

public class ListViewExemple extends Application {
    public void start(Stage primaryStage) {
        primaryStage.setTitle("List View Exemple");
        StackPane root = new StackPane();

        //Création de l'ObservableList
        ObservableList<String> names = FXCollections.observableArrayList("Adam",
            "Alex", "Alfred", "Albert",
            "Brenda", "Connie", "Derek", "Donny",
            "Lynne", "Myrtle", "Rose", "Rudolph",
            "Tony", "Trudy", "Williams", "Zach");

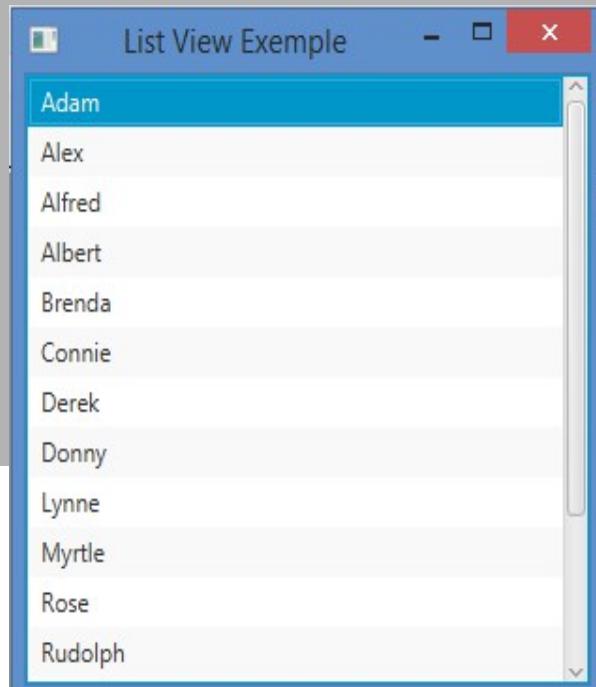
        //Création de la ListView
        ListView<String> listView = new ListView<String>(names);

        listView.setEditable(true);

        root.getChildren().add(listView);
        primaryStage.setScene(new Scene(root, 300,
280));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

L'exécution de ce programme affiche la fenêtre ci-contre.



Autre manière pour remplir la ListView

//Création de la ListView

```
ListView<String> listView = new  
    ListView<String>(names);  
    names.addAll("Adam", "Alex", "Alfred", "Albert", "Brenda", "Connie",  
        "Derek", "Donny", "Lynne", "Myrtle", "Rose", "Rudolph",  
        "Tony", "Trudy", "Williams", "Zach" );
```

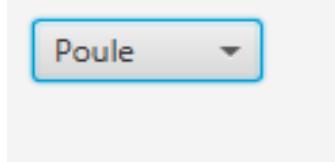
Les ChoiceBox

Une ChoiceBox est une liste déroulante qui permet à l'utilisateur d'effectuer un choix.

15^{ème} exemple

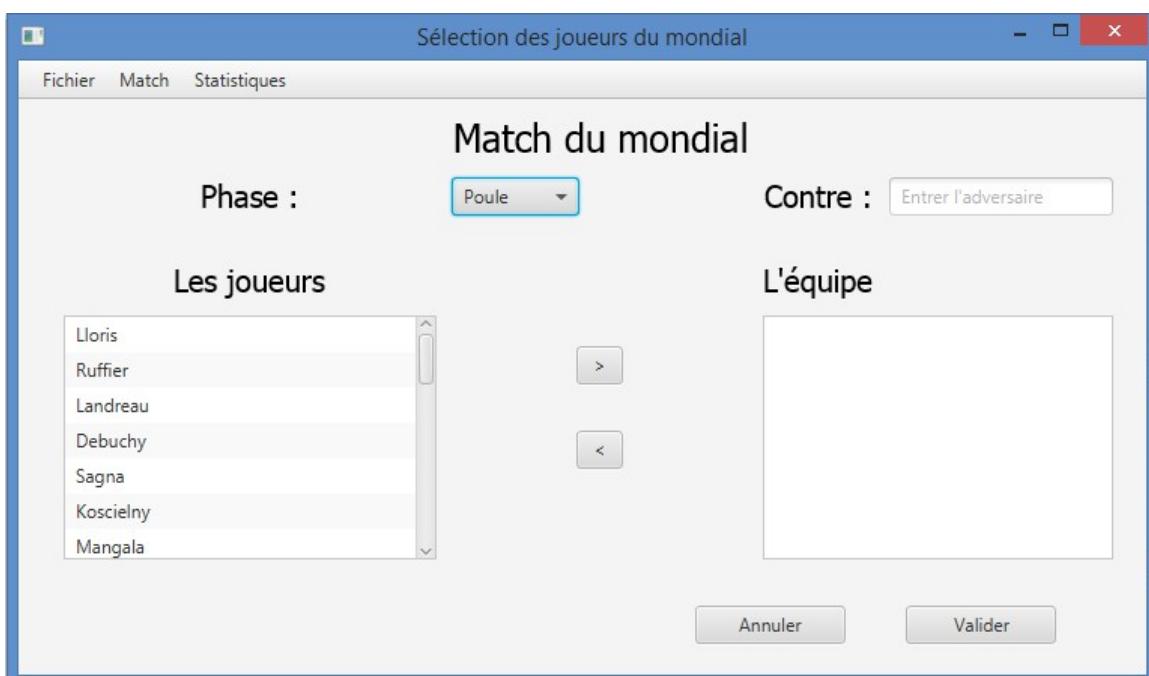
```
ChoiceBox<String> choix = new ChoiceBox<String>();  
choix.setItems(FXCollections.observableArrayList("Poule", "1/8 Finale", "1/4 Finale",  
    "1/2 Finale", "Finale"));  
choix.getSelectionModel().select(0);
```

Cela donne à l'écran :



Exercice récapitulatif : création d'une l'IHM

Un informaticien amateur de football décide de créer son application pour gérer les matchs de la coupe du monde. On donne ici un exemple d'une interface utilisateur réalisée.



Le menu Ficher est classique, voir le 11^{ème} exemple.

Le menu Match propose un sous-menu radio: Poule, 1/8 Finale, ¼ Finale, ½ Finale et Finale.

Le sous-choix Poule liste les matchs de poules, le sous-choix 1/8 Finale liste le 8^{ème} de finale, le sous-choix 1/4 Finale liste le quart...

Le menu Statistiques affiche des statistiques non étudiées ici.

On peut utiliser un **BorderPane** dans lequel on place

- le menu en haut,
- un GridPane au milieu,
- une Hbox en bas pour les 2 boutons Annuler et Valider.

La ListView de gauche contient la liste des joueurs retenus pour le mondial.

```
String nomsdesjoueurs[] = {"Lloris", "Ruffier", "Landreau", "Debuchy", "Sagna", "Koscielny",
                            "Mangala", "Sakho", "Varane", "Evra", "Digne",
                            "Cabaye", "Pogba", "Matuidi", "Mavuba", "Sissoko",
                            "Schneiderlin", "Valbuena",
                            "Benzema", "Cabella", "Giroud", "Griezmann", "Rémy"};
```

```
ObservableList<String> lesnoms = FXCollections.observableArrayList(nomsdesjoueurs);
ListView<String> listView = new ListView<String>(lesnoms);
```

La **ListView** de droite est vide au lancement de l'application.

Le thread principal Java FX et la classe Application

Java FX crée un thread principal pour exécuter la méthode **start()** de l'application, gérer les événements et exécuter les animations. Comme pour tous les exemples présentés, la création de l'objet Scene et de tous les objets inclus doit être faite dans le thread principal.

Une application Java FX se termine en cliquant sur la croix (x) de la dernière fenêtre ouverte ou en exécutant la méthode **Platform.exit()**. Cette fermeture d'une application Java FX respecte le cycle de vie d'une application Java FX et est préférable à l'utilisation de la méthode **System.exit(int)**.

La méthode **main()** est nécessaire pour exécuter l'application Java FX en ligne de commande. Elle contient la méthode **lauch()** qui lance l'exécution de l'application.

Gestion des événements

Les événements

Les événements signalent l'arrivée de quelque chose que l'application doit traiter : l'utilisateur clique avec sa souris, déplace la souris avec le bouton gauche enfoncé, appuie sur une touche particulière...

Les événements Java FX sont des objets instances de classes qui dérivent toutes de la classe **Event**.

Voici un extrait de la documentation Java FX qui présente les sous classes de **Event**:

Class Event

Direct Known Subclasses:

[ActionEvent](#), [CheckBoxTreeItem.TreeModificationEvent](#), [InputEvent](#), [ListView.EditEvent](#),

```
MediaErrorEvent, ScrollToEvent, SortEvent, TableColumn.CellEditEvent,  
TransformChangedEvent, TreeItem.TreeModificationEvent,  
TreeTableColumn.CellEditEvent, TreeTableView.EditEvent, TreeView.EditEvent,  
WebErrorEvent, WebEvent, WindowEvent, WorkerStateEvent
```

Si on prend **InputEvent**, voici un extrait de la documentation Java FX qui présente ses sous classes:

Class InputEvent

Direct Known Subclasses:

```
ContextMenuEvent, DragEvent, GestureEvent, InputMethodEvent, KeyEvent,  
MouseEvent, TouchEvent
```

MouseEvent est la classe de l'événement provoqué en cliquant sur un des boutons de la souris.

Java FX définit des méthodes pratiques (convenience method) pour installer la réponse à l'apparition (l'occurrence) d'un événement.

Presque toutes ces méthodes sont définies dans la classe **Node**.

Ces méthodes sont utilisées pour installer les gestionnaires d'événement **EventHandler**.

Ces méthodes sont spécialisées : elles ne peuvent installer que le gestionnaire (**EventHandler**) pour l'événement considéré ou au plus pour un événement du type d'une des supers classe (classes mères) de l'événement considéré.

Le gestionnaire de l'événement

Le gestionnaire de l'événement est toujours une instance d'une classe qui implémente l'interface EventHandler.

EventHandler est une interface générique paramétrée :

```
interface EventHandler<T extends Event>
```

c'est à dire qu'elle est utilisable pour tous les événements de classe **T** de type **Event** et bien sûr de ses dérivés. Il faut donc juste préciser la classe de l'événement traité lors de la création du gestionnaire de type **EventHanler**.

Exemple de création d'un gestionnaire qui traite l'événement **MouseEvent**:

```
class Gestionnaire implements EventHandler<MouseEvent> {  
    ....  
}
```

L'interface **EventHandler** ne contient qu'une seule méthode

```
void handle(T event)
```

que le gestionnaire de l'événement doit définir, **event** étant l'événement survenu (event est un objet de type T, classe de l'événement traité).

Voici le gestionnaire complet qui traite l'événement **MouseEvent** avec la définition de la méthode **handle** :

```
class Gestionnaire implements EventHandler<MouseEvent> {  
    public void handle(MouseEvent event) {  
        System.out.println("Vous avez cliqué");  
    }  
}
```

Une interface qui ne contient qu'une seule méthode est appelée une interface fonctionnelle.

Les méthodes pour installer le gestionnaire de l'événement (dans la classe Node)

Le format de ces méthodes est le suivant :

setOnEvent-type(EventHandler<? super event-class> value)

- **Event-type** précise le type de l'événement,
- **EventHandler** précise que le gestionnaire de l'événement est de type **EventHandler**. **EventHandler** est une interface définie par Java FX, voir ci-dessus.
- **<? super event-class>** indique que la méthode accepte un gestionnaire d'événement pour **event-class** ou pour une de ses super classes.
- **Value** est l'objet de type **EventHandler** qui traite l'événement.

Par exemple, la méthode suivante pour l'événement «on clique avec la souris» est définie dans la classe **Node** :

setOnMouseClicked(EventHandler<? super MouseEvent> value)

- **Event-type = MouseClicked** précise le type de l'événement,
- **<? super MouseEvent>** indique que la méthode accepte un gestionnaire d'événement pour **MouseEvent** ou pour une des ses super classes.
- **value** est l'objet de type **EventHandler** qui traite l'événement.

Le seul paramètre modifiable est **value**, c'est à dire l'objet de type **EventHandler** qui traite l'événement. On a vu que le gestionnaire de l'événement est toujours une instance d'une classe qui implémente l'interface **EventHandler**.

Exemple où on associe un objet composant avec le gestionnaire précédent quand on clique avec la souris :

```
composant.setOnMouseClicked(new Gestionnaire());
```

Etude détaillée d'un exemple

L'exemple qui suit reprend le code du 2^{ème} exemple : le décors est vide, on associe l'événement « clic souris » quand on clique dans la fenêtre avec une classe Gestionnaire qui implémente l'interface EventHandler.

16^{ème} exemple

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class BorderPaneEx1 extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Titre de la fenêtre
        primaryStage.setTitle("Un exemple de BorderPane");
        // Création de la racine - decors -
```

```

        BorderPane root = new BorderPane() ;
        //Installation du gestionnaire
        root.setOnMouseClicked(new Gestionnaire());
        // Création de la scène, association du décors et de la scène, taille de la scène
        Scene scene = new Scene(root,400,300);

        // On place la scène dans la fenêtre
        primaryStage.setScene(scene);
        // On affiche la fenêtre
        primaryStage.show();
    }

    class Gestionnaire implements EventHandler<MouseEvent> {
        public void handle(MouseEvent event) {
            // TODO Auto-generated method stub
            System.out.println("Vous avez cliqué");
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

La classe Gestionnaire est déclarée à l'intérieur de la classe BorderPaneEx1. Cela permet d'accéder dans Gestionnaire aux attributs de BorderPaneEx1. Cette déclaration interne fait que Gestionnaire est appelée une **classe interne - inner class -**.

La ligne

```
class Gestionnaire implements EventHandler<MouseEvent>
```

montre bien que **Gestionnaire** implémente **EventHandler** en précisant le type de la classe de l'événement traité, ici **MouseEvent**.

La méthode **handle()** est bien définie dans Gestionnaire.

La ligne

```
root.setOnMouseClicked(new Gestionnaire());
```

installe un **objet de type Gestionnaire** pour traiter les clics de souris.

Cet objet est **anonyme**. On pourrait le créer à part et lui donner un identifiant mais cela ne se fait pas car cela ne sert (pratiquement) jamais.

Une classe interne anonyme pour traiter l'événement

Exemple simple de classe interne anonyme :

```

interface Animal {
    public void parler();
}

public class TestAnonyme {

//1ère classe anonyme
    Animal lion = new Animal() {

```

```

        public void parler() {
            System.out.println("Je rugis comme un lion");
        }
    };
//2ème classe anonyme
Animal chien = new Animal() {
    public void parler() {
        System.out.println("J'aboie comme un chien");
    }
};

public static void main(String[] args) {
    TestAnonyme t = new TestAnonyme();
    t.lion.parler();
    t.chien.parler();
}
}

```

L'interface Animal contient 1 seule méthode, la méthode parler().

L'objet lion est instancié à partir de l'interface Animal, mais cela peut se faire aussi à partir d'une classe.

L'objet lion est construit comme un objet normal avec l'opérateur **new** suivi du constructeur. Le bloc de code qui suit les parenthèses du constructeur et placé entre les accolades définit ici complètement la classe anonyme, c'est à dire ici la définition de la méthode parler().

La remarque du paragraphe précédent nous amène à créer une classe anonyme à la place de la classe Gestionnaire. C'est pratiquement toujours ainsi que les programmeurs Java traitent les événements.

```

root.setOnMouseClicked(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent arg0) {
        System.out.println("Vous avez cliqué");
    }
});

```

On crée un objet anonyme de type EventHandler avec **new EventHandler()**.

On précise ensuite la définition de la classe anonyme entre les accolades, ici en grisé

```

root.setOnMouseClicked(new EventHandler<MouseEvent>() {
    public void handle( MouseEvent arg0) {
        System.out.println("Vous avez cliqué");
    }
});

```

La définition de cette classe anonyme ne comporte que la définition de la seule méthode **handle()** de l'interface **EventHandler**.

17^{ème} exemple avec une classe anonyme

```

public void start(Stage primaryStage) {

    primaryStage.setTitle("Un exemple de BorderPane");
    BorderPane root = new BorderPane();

```

```

root.setOnMouseClicked(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent arg0) {
        System.out.println("Vous avez cliqué");
    }
});

Scene scene = new Scene(root,400,300);
primaryStage.setScene(scene);
primaryStage.show();
}

```

L'argument, ici **arg0**, de type **MouseEvent** de la méthode **handle()** contient des informations sur l'événement survenu.

18^{ème} exemple : Affichage du bouton cliqué

```

root.setOnMouseClicked(new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        System.out.println("Vous avez cliqué");

        MouseButton b = arg0.getButton();
        System.out.println(b);
        //OU System.out.println(b.name());
        //OU System.out.println(arg0.getButton());
    }
});

```

L'affichage donne **PRIMARY** pour un clic avec le bouton de gauche, **MIDDLE** pour celui du milieu et **SECONDARY** pour le bouton de droite.

19^{ème} exemple : Affichage des coordonnées du curseur

```

public void handle(MouseEvent arg0) {
    // Coordonnées relatives du clic par-rapport à la source de l'événement
    System.out.println("X : "+arg0.getX());
    System.out.println("Y : "+arg0.getY());
    // Coordonnées relatives du clic par-rapport à l'écran
    System.out.println("Dans l'écran X : "+arg0.getScreenX());
    System.out.println("Dans l'écran Y : "+arg0.getScreenY());
    // Coordonnées relatives du clic par-rapport à la scène
    System.out.println("Dans la scène X : "+arg0.getSceneX());
    System.out.println("Dans la scène Y : "+arg0.getSceneY());
}

```

20^{ème} exemple : Affichage du nombre de clic

```

public void handle(MouseEvent arg0) {
    System.out.println("Nombre de clic : "+arg0.getClickCount());
    ...
}

```

L'événement OnAction et la classe abstraite ButtonBase

L'événement **OnAction** dérive comme tous les autres événements de la classe **Event**.

La classe abstraite **ButtonBase** déclare la méthode **setOnAction** utilisée pour installer le gestionnaire de l'événement **OnAction**.

Les classes **Button**, **CheckBox**, **Hyperlink**, **MenuButton** et **ToggleButton** dérivent de la classe **ButtonBase**.

Il est avantageux pour ces classes d'installer un gestionnaire de l'événement **OnAction**.

21^{ème} exemple, voir le 7^{ème} exemple: fenêtre d'identification avec demande de login et de mot de passe

```
PasswordField pwBox = new PasswordField();
TextField userTextField = new TextField();
Text message = new Text("N'oubliez pas de valider!");

Button btn = new Button("Validez");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        String nom = userTextField.getText();
        String mp = pwBox.getText();
        if (nom.length()!=0 && mp.length()!=0)
            System.out.println("Nom:"+nom+" Mdp: "+mp);
        else {
            message.setFill(Color.FIREBRICK);
            message.setText("N'oubliez pas de remplir les 2 champs!");
        }
    }
});
```

Exercice récapitulatif : gestion des événements

On reprend l'exercice précédent en ajoutant la gestion des événements.

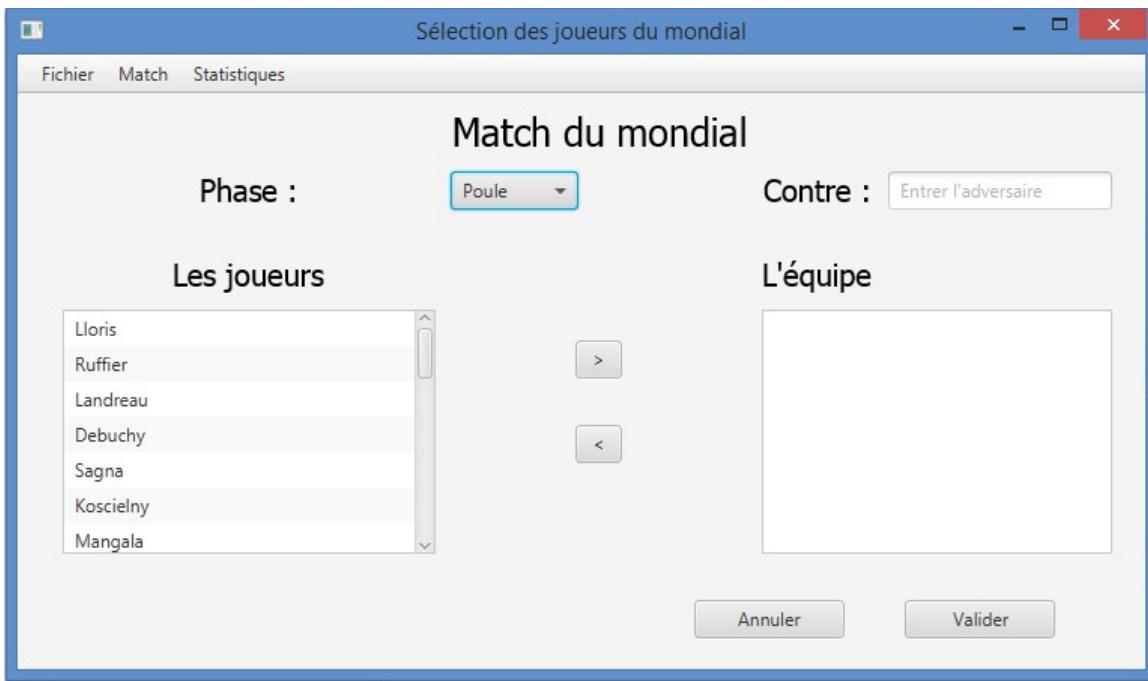
On constitue l'équipe avec le bouton **>**. On sélectionne un joueur dans la ListView de gauche «Les joueurs» puis on clique sur ce bouton et le nom du joueur est déplacé dans la ListView de droite «L'équipe».

On enlève un joueur de l'équipe en cliquant sur le bouton **<**. Le nom du joueur choisi est alors déplacé de la ListView «L'équipe» vers la ListView «Les joueurs».

Le bouton **Annuler** permet de revenir à la situation initiale, les 23 joueurs dans la ListView de gauche et la ListView de droite vide.

Le bouton **Valider** affiche dans la console la phase concernée, l'adversaire et la sélection.

La sortie de l'application peut se faire par Fichier/Sortir. La méthode invoquée alors doit être Platform.exit() pour mettre fin uniquement à l'application Java FX.



Bonus

Le menu Match propose: Poule, 1/8 Finale, ¼ Finale, ½ Finale et Finale.

Le sous-choix Poule liste les maths de la poule, le sous-choix 1/8 Finale liste le 8^{ème} de finale, le sous-choix ¼ Finale liste le quart...

Choisir un match et afficher la liste des joueurs sélectionnés.

Les expressions Lambda

Les expressions **Lambda** sont un nouveau concept apparu avec Java 8.

Les expressions **Lambda** s'appliquent aux interfaces ne contenant qu'une seule méthode à définir. Une interface contenant une seule méthode est appelée une interface fonctionnelle.

L'utilisation des expressions **Lambda** est une 1^{ère} étape dans Java de la **programmation dite fonctionnelle**.

22^{ème} exemple : expression Lambda avec un gestionnaire d'événement

L'installation du gestionnaire suivant

```
Button btn = new Button(" OK ");
btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        System.out.println("Vous avez validé");
    }
});
```

peut être remplacé par l'expression Lambda écrite en gras

```
Button btn = new Button(" OK ");
btn.setOnAction(
    (ActionEvent e) -> System.out.println("Vous avez validé")
);
```

ou même par

```
Button btn = new Button(" OK ");
btn.setOnAction(
    (e) -> System.out.println("Vous avez validé")
);
```

(ActionEvent e) -> System.out.println("Vous avez validé") est une expression Lambda.

Il n'est pas nécessaire de préciser le type du paramètre (ActionEvent) car le compilateur le déduit du contexte.

Attention: par de point-virgule si il n'y a qu'une seule instruction.

Si il y a plusieurs instructions à exécuter, il faut les placer dans un bloc :

```
btn.setOnAction(
    (ActionEvent e)-> {
        String nom = userTextField.getText();
        String mp = pwBox.getText() ;
        if (nom.length()!=0 && mp.length()!=0)
            System.out.println("Nom:"+nom+" Mdp: "+mp);
        else {
            message.setFill(Color.FIREBRICK);
            message.setText("N'oubliez pas de remplir les 2 champs!!!");
        }
    }
);
```

Syntaxe d'une expression Lambda sans paramètre

Cas où la méthode pour installer l'interface de gestion est du type :

setOnEvenement()

L'expression Lambda correspondante est

() → System.out.println("Lambda sans paramètre")

Syntaxe d'une expression Lambda avec un paramètre

Cas où la méthode pour installer l'interface de gestion est du type :

setOnEvenement(param)

L'expression Lambda correspondante est

(param) → System.out.println("Lambda avec un paramètre"+ param)

Syntaxe d'une expression Lambda avec plusieurs paramètres

Cas où la méthode pour installer l'interface de gestion est du type :

setOnEvenement(p1,p2)

L'expression Lambda correspondante est

(p1, p2) → System.out.println("Lambda avec 2 paramètres"+ p1 + ", "+p2)

Nécessité de préciser le type

Le compilateur peut quelque fois demander le type du paramètre, solution :

(Type p) → System.out.println("Lambda avec un paramètre typé "+ p.getName())

Corps d'une expression Lambda

C'est nécessaire quand plusieurs instructions doivent être exécutées (voir le 22^{ème} exemple).

Avec une seule instruction :

```
( a , b ) → System.out.println("Une expression Lambda")
```

Avec plusieurs instructions :

```
( a , b ) → {  
    System.out.println("Une première instruction dans l'expression Lambda") ;  
    System.out.println("Une deuxième instruction dans l'expression Lambda") ;  
}
```

Valeur retournée par une expression Lambda

On peut ajouter l'instruction **return** dans le corps.

```
( a , b ) → {  
    System.out.println("Une première instruction dans l'expression Lambda") ;  
    return "texte retourné dans l'expression Lambda" ;  
}
```

Dans le cas d'une seule instruction :

```
( a , b ) → {  
    return a > b ;  
}
```

ou même

```
( a , b ) → a > b
```

Les expressions Lambda et les objets

Une expression Lambda est un **objet**. On peut affecter une expression Lambda à une variable et l'utiliser.

```
public interface MonComparateur {  
    public boolean compare(int a, int b) ;  
}
```

```
MonComparateur moncomparateur = (a1 , a2 ) → a1 > a2 ;  
boolean resultat = moncomparateur.compare(2,5) ;
```

Le programme suivant utilise l'exemple précédent :

```
interface MonComparateur {  
    public boolean compare(int a, int b);  
}  
  
public class Lambda1 {  
    public static void main(String[] args) {  
  
        MonComparateur moncomparateur = ( a1 , a2 ) -> a1>a2;  
  
        /* OU BIEN  
           MonComparateur moncomparateur = ( a1 , a2 ) -> {  
               return a1>a2;  
           } ;  
        */
```

```

        boolean resultat = moncomparateur.compare(2,5);
        System.out.println("resultat = "+resultat);
    }
}

```

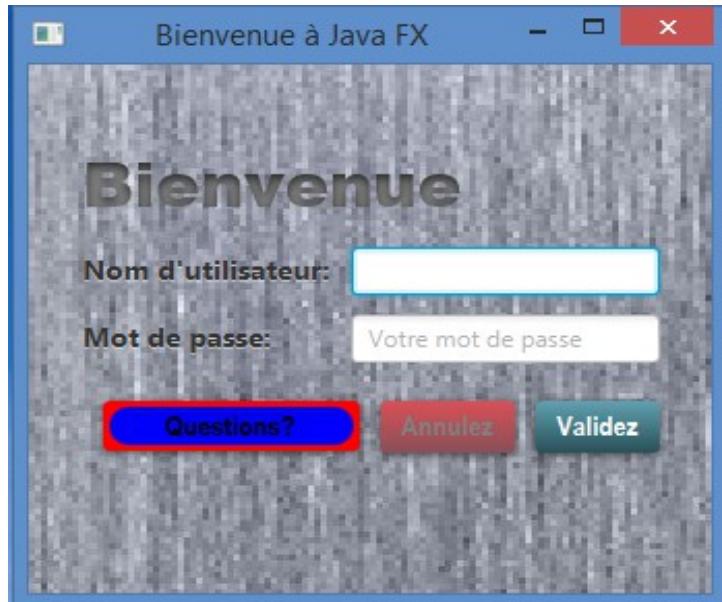
Exercice

Reprendre l'exercice précédent sur la sélection des joueurs en utilisant les expressions Lambda.

Java FX CSS

Ce cours ne se veut pas exhaustif sur l'utilisation de Java FX CSS, on ne présente ici que quelques exemples, voir le lien pour des informations complètes <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.htm>

L'exemple ci-contre présente une IHM personnalisée avec une image de fond, des labels utilisant une police particulière, des boutons particuliers.



La feuille de style

Eclipse crée une feuille de style **application.css** dans le package de travail.

On indique à Java FX l'utilisation de cette feuille de style pour toute la scène de la manière suivante :

```
scene.getStylesheets().add("application/application.css");
```

Cela suppose un package de nom **application**.

Classes et ID CSS

Java FX définit des classes CSS de base ou *Style class* pour la plupart des **nodes** : bouton, label... Par exemple, la classe CSS de base *Style class* est *button* pour un objet de type **Button**, la classe CSS de base est *label* pour un objet de type **Label**.

On peut également affecter une nouvelle classe CSS à un objet Java FX ou bien l'identifier par un ID.

Il est simple d'utiliser la classe CSS *button* si tous les boutons doivent avoir la même apparence. Maintenant, si un ou plusieurs boutons ont une apparence singulière, il faut que chacun de ces

boutons se voit attribuer une classe CSS spécifique ou un ID particulier.

Les classes de base CSS ou *Style class*

Java FX désigne les classes de base CSS par le terme de *Style class*.

Un certain nombre de nœuds ont leur propre *Style class*.

La scène – objet de type **Scene** – a comme Style class *root*.

Un objet de type **Button** a comme Style class *button*.

Un objet de type **Label** a comme Style class *label*.

Un objet de type **Button** a comme Style class *button*.

Un objet de type **ImageView** a comme Style class *image-view*.

Un objet de type **CheckBox** a comme Style class *check-box*.

...

Voir le site <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>

Java FX suit les normes du CSS. A quelques rares exceptions près, les noms des propriétés Java FX CSS sont préfixées par **-fx-**.

Explication sur le bouton *Questions?*

L'attribution de l'ID *questions* est faite dans le code Java de la façon suivante :

```
Button qubutton = new Button("Questions?");  
qubutton.setId("questions");
```

Dans la feuille de style

```
#questions {  
    -fx-background-color: red, blue ;  
    -fx-background-insets: 0,3;  
    -fx-background-radius: 3,30;  
    -fx-padding: 5 30 5 30;  
    -fx-text-fill: black;  
    -fx-font-size: 14px;  
}
```

#questions signifie que l'ID *questions* a été attribué au bouton.

-fx-background-color: red, blue ; le bouton est coloré en rouge puis par-dessus en bleu.

-fx-background-insets: 0,3; le bouton est entièrement coloré en rouge mais par-dessus en bleu dans un cadre plus petit de 3 pixels de chaque coté.

-fx-background-radius: 3,30; le cadre rouge a ses angles arrondis avec un rayon de 3 pixels alors que le cadre bleu a ses angles arrondis avec un rayon de 30 pixels.

-fx-padding: 5 30 5 30; le padding définit une marge autour du texte, dans l'ordre :haut, droite, bas, gauche.

-fx-text-fill: black; la couleur du texte.

-fx-font-size: 14px; la taille de la police.

Les pseudo-classes

Java FX CSS accepte notamment les pseudo-classes suivantes:

- **focused** pour un élément qui a le focus,
- **hover** pour un élément survolé par la souris,
- **pressed** quand un élément est cliqué.

Par exemple: le bouton Questions? pour lequel on diminue le rectangle bleu interne quand il a le focus.

```
#questions:focused {  
    -fx-background-color: red, blue;  
    -fx-background-insets: 0,6;  
    -fx-background-radius: 3,40;  
}
```

21^{ème} exemple

Code Java de l'IHM précédent.

```
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.stage.Stage;  
import javafx.scene.Scene;  
import javafx.scene.control.*;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.text.Text;  
  
public class Style1 extends Application {  
    @Override  
    public void start(Stage primaryStage) {  
        GridPane grid = new GridPane();  
        Scene scene = new Scene(grid, 330, 275);  
  
        grid.setAlignment(Pos.CENTER);  
        grid.setHgap(10);  
        grid.setVgap(10);  
        grid.setPadding(new Insets(25, 25, 25, 25));  
        Text scenetitle = new Text("Bienvenue");  
        // ID CSS «bienvenue» pour le Text scenetitle  
        scenetitle.setId("bienvenue");  
        grid.add(scenetitle, 0, 0, 2, 1);  
  
        Label userName = new Label("Nom d'utilisateur:");  
        grid.add(userName, 0, 1);  
        TextField userTextField = new TextField();  
        userTextField.setPromptText("Votre nom");  
        grid.add(userTextField, 1, 1);  
  
        Label pw = new Label("Mot de passe:");  
        grid.add(pw, 0, 2);  
  
        scene.getStylesheets().add("application/application.css");  
  
        PasswordField pwBox = new PasswordField();  
        pwBox.setPromptText("Votre mot de passe");  
        grid.add(pwBox, 1, 2);
```

```

        Button valid = new Button("Validez");
        // classe CSS « button1 » pour le bouton valid
        valid.getStyleClass().add("button1");

        Button annul = new Button("Annulez");
        Button qubutton = new Button("Questions ?");
        // ID CSS « questions » pour le bouton qubutton
        qubutton.setId("questions");
        HBox hbBtn = new HBox(10);
        hbBtn.setAlignment(Pos.BOTTOM_RIGHT);

        hbBtn.getChildren().addAll(qubutton, annul, valid);

        grid.add(hbBtn, 0, 4, 2, 1);
        final Text actiontarget = new Text();
        actiontarget.setId("message");

        grid.add(actiontarget, 1, 6);

        valid.setOnAction((ev) -> {
            String nom = userTextField.getText();
            String pwd = pwBox.getText();
            if (nom.isEmpty() || pwd.isEmpty())
                actiontarget.setText("Champs incomplets");
        });
        annul.setOnAction((ev)-> {
            userTextField.setText("");
            pwBox.setText("");
            actiontarget.setText("");
        });

        primaryStage.setTitle("Bienvenue à Java FX");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

Feuille CSS de l'exemple précédent

La Style class de l'objet Scene est désigné par le nom *root*. L'image **fond.jpg** est dans le dossier package, ici application.

```

.root {
    -fx-background-image: url("fond.jpg");
}
.label {

```

```

        -fx-font-size: 14px;
        -fx-font-weight: bold;
        -fx-text-fill: #333333;
        -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
    }

#questions {
    -fx-background-color: red, blue ;
    -fx-background-insets: 0,3;
    -fx-background-radius: 3,30;
    -fx-padding: 5 30 5 30;
    -fx-text-fill: black;
    -fx-font-size: 14px;
}

}

#questions:focused {
    -fx-background-color: red, blue;
    -fx-background-insets: 0,6;
    -fx-background-radius: 3,40;
}

}

.button {
    -fx-text-fill: grey;
    -fx-font-size: 14px;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#DA5058, #8A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
}

}

.button:hover{
    -fx-background-color: #AA5050;
}

}

.button:focused {
    -fx-background-color: -fx-faint-focus-color, -fx-focus-color, -fx-inner-border, -fx-body-color;
    -fx-background-insets: -2, -0.3, 1, 2;
    -fx-background-radius: 7, 6, 4, 3;
}

}

.button1 {
    -fx-text-fill: white;
    -fx-font-size: 14px;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
}

}

.button1:hover{
    -fx-background-color: #395bae;
}

}

#bienvenue {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6, 0.0 , 0 , 2 );
}

```

```
}
```

```
#message {
```

```
-fx-fill: FIREBRICK;
```

```
-fx-font-weight: bold;
```

```
-fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
```

```
}
```

Voir le lien <http://fxexperience.com/2011/12/styling-fx-buttons-with-css/> qui présente un grand nombre de boutons avec tous des styles différents.

Exercice

Tester cet exemple.

Etudier le fichier CSS. Modifier des paramètres, observer et justifier les changements visuels obtenus.