Introduction à JavaFX

JavaFX est une technologie créée par Sun Microsystems qui appartient désormais à Oracle.

Avec l'apparition de Java 8 en mars 2014, JavaFX devient la bibliothèque de création d'interface graphique officielle du langage Java, pour toutes les sortes d'application (applications mobiles, applications sur poste de travail, applications Web), le développement de son prédécesseur Swing étant abandonné (sauf pour les corrections de bogues).

JavaFX contient des outils très divers, notamment pour les médias audio et vidéo, le graphisme 2D et 3D, la programmation Web, la programmation multi-fils etc.

Le SDK de JavaFX étant désormais intégré au JDK standard Java SE,

Main
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 :

- Construit une instance de la classe Application spécifiée
- Appelle la méthode init ()
- Appelle la méthode start (javafx.stage.Stage)
- Attend l'achèvement de l'application,
- Appelle la méthode stop ()

Notez que la méthode de démarrage (start) est abstraite et doit être surchargée. Les méthodes d'initialisation (init) et d'arrêt (stop) ont des implémentations concrètes qui ne font rien.

L'exécution du code suivant montre que l'exécution de la méthode init n'est pas faite par le même fil d'exécution (thread) que l'appel au constructeur par défaut et l'exécution de la méthode start.

```
public class MainJavaFX extends Application {
	private Button button;
	private StackPane root;
	private Scene scene;

	public MainJavaFX() {
		System.out.print(Thread.currentThread().getName());
		System.out.println(" appelle le constructeur de la Classe MainJavaFX");

		button = new Button("Salut le monde !");
		button.setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event) {
				System.out.println("Hello World!");
			}}	  );

		// Nœud racine.
		root = new StackPane();
		root.getChildren().setAll(button);
		// Configuration de la scène.
		scene = new Scene(root);
	}
```

```
	@Override
	public void init() throws Exception {
		super.init();
		// Faire des initialisations ici.
		System.out.print(Thread.currentThread().getName());
		System.out.println("execute la méthode init de la classe MainJavaFX");
	}

	@Override
	public void start(Stage primaryStage) {
		System.out.print(Thread.currentThread().getName());
		System.out.println(" execute la méthode start de la classe MainJavaFX");
		// Configuration de la fenêtre.
		primaryStage.setScene(scene);
		primaryStage.setTitle("Ma première application");
		primaryStage.setWidth(350);
		primaryStage.setHeight(300);
		primaryStage.show();
	}

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

Layout – Gestionnaire de la mise en page

Un *layout* ou gestionnaire de mise en page est un nœud graphique qui hérite de la classe javafx.scene.layout.Pane. Il s'agit d'une entité qui contient d'autres nœuds et qui est chargée de les déplacer, de les disposer, voire de les redimensionner de manière à changer la présentation de cet ensemble de nœuds et à les rendre utilisables dans une interface graphique.

Extrait de la page https://java.developpez.com/faq/javafx/?page=Mise-en-page qui contient des exemples et informations sur les *layouts* de JavaFX.
Les *layout* fonctionnent donc de manière similaire aux *layouts* de AWT/Swing à ceci près qu'il ne s'agit pas ici de classes utilitaires, mais d'instances de la classe Region qui sont directement placées dans la scène. Un gestionnaire est un nœud graphique et peut donc être positionné et manipulé, ou subir des transformations et effets comme n'importe quel autre nœud de SceneGraph.

Les différences importantes entre Panel (panneau) et Group (groupe) sont :
• Un panneau peut avoir sa propre taille et être redimensionnable, par contre un groupe prend les limites collectives de ses enfants et n'est pas directement redimensionnable.
• Le panneau doit être utilisé lorsque vous souhaitez positionner ses noeuds en position absolue.

Un layout (organisation) est une classe de l'API Graphique permettant d'organiser les objets graphiques dans la fenêtre. En JavaFX, il existe huit types de layouts :

1. Le **BorderPane** qui vous permet de diviser une zone graphique en cinq parties : top, down, right, left et center.
2. La **Hbox** qui vous permet d'aligner horizontalement vos éléments graphiques.
3. La **VBox** qui vous permet d'aligner verticalement vos éléments graphiques.
4. Le **StackPane** qui vous permet de ranger vos éléments de façon à ce que chaque nouvel élément inséré apparaisse au-dessus de tous les autres.
5. Le **GridPane** permet de créer une grille d'éléments organisés en lignes et en colonnes

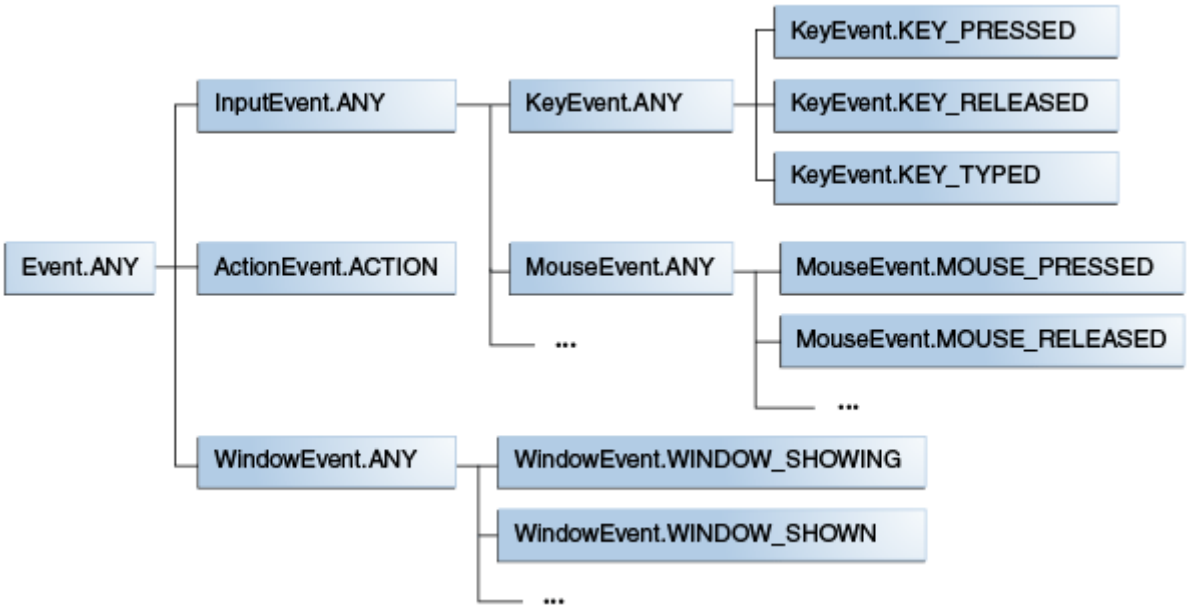
6. Le **FlowPane** permet de ranger des éléments de façon à ce qu'ils se positionnent automatiquement en fonction de leur taille et de celle du layout.
7. Le **TilePane** est similaire au FlowPane, chacune de ses cellules fait la même taille.
8. L'**AnchorPane** permet de fixer un élément graphique par rapport à un des bords de la fenêtre : top, bottom, right et left.

Les évènements – source : documentation oracle

Dans les applications JavaFX, les événements signifient que quelque chose s'est produit : un utilisateur a cliqué sur un bouton, appuie sur une touche, déplace une souris ou effectue d'autres actions, …
Dans JavaFX, un événement est une instance de la classe `javafx.event.Event` ou de toute sous-classe d'`Event`.
JavaFX fournit plusieurs sous-classe d'`Event`, notamment `DragEvent, KeyEvent, MouseEvent, ScrollEvent` et d'autres.
Vous pouvez définir votre propre sous-classe d'`Event` en étendant la classe Event.

Chaque événement inclut les informations suivantes : Event Type, Source and Target.

La hiérarchie des évènements :



Le type d'événement de niveau supérieur dans la hiérarchie est `Event.ROOT`, qui est équivalent à `Event.ANY`.
Dans les sous-types, le type d'événement ANY est utilisé pour désigner n'importe quel type d'événement dans la classe d'événements. Par exemple, pour fournir la même réponse à n'importe quel type d'événement de type « touche », utilisez `KeyEvent.ANY` comme type d'événement pour le filtre d'événements ou le gestionnaire d'événements.
Pour répondre uniquement lorsque la touche est libérée, utilisez le type d'événement `KeyEvent.KEY_RELEASED` pour le filtre ou le gestionnaire

Les gestionnaires d'événements (`EventHandler`) vous permettent de gérer les événements.
Un nœud peut avoir un ou plusieurs gestionnaires pour gérer un événement.
Un seul gestionnaire peut être utilisé pour plusieurs nœuds et plusieurs types d'événements.

Enregistrement d'un gestionnaire d'événements

Pour traiter un événement, un nœud doit enregistrer un gestionnaire d'événements. Un gestionnaire d'événements est une implémentation de l'interface `EventHandler`. La méthode `handle ()` de cette interface fournit le code qui est exécuté lorsque l'événement associé au gestionnaire est reçu par le noeud qui a enregistré le gestionnaire.
Pour enregistrer un gestionnaire, utilisez la méthode `addEventHandler` de la classe `javafx.scene.Node`. Cette méthode prend le type d'événement et le gestionnaire comme arguments :

```
public final <T extends Event> void addEventHandler(EventType<T> eventType,
                                    EventHandler<? super T> eventHandler).
```

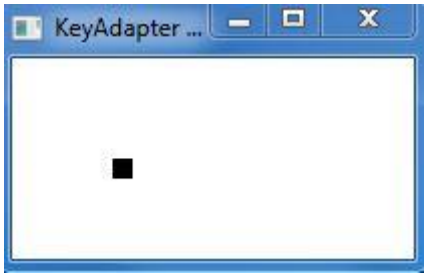
```
// Register the same handler for two different nodes
myNode1.addEventHandler(DragEvent.DRAG_EXITED, handler);
myNode2.addEventHandler(DragEvent.DRAG_EXITED, handler);

// Register the handler for another event type
myNode1.addEventHandler(MouseEvent.MOUSE_DRAGGED, handler);
```

La méthode pratique d'enregistrement d'un seul gestionnaire de type « `ActionEvent` » est: `setOnAction(EventHandler<ActionEvent> value).`

Correction de l'exercice « Beep » du cours d'Arnaud Pêcher en utilisant API JavaFX

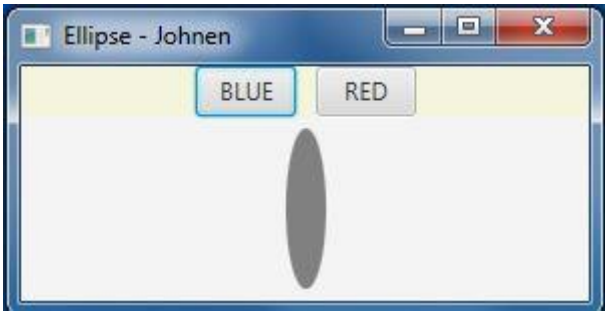

```
/*
 *  Prof. Janice T. Searleman, University of clarkson
 *  modified by C. Johnen
 */

public class Beeper extends Application implements  EventHandler<ActionEvent> {
	StackPane root ;

	public Beeper() { root = new StackPane();    }

	@Override
	public void start(Stage primaryStage) {
		primaryStage.setTitle("Beeper - Johnen");
		Scene scene = new Scene(root,200,100);
		primaryStage.setScene(scene);
		primaryStage.show();   }

	@Override
	public void handle(ActionEvent e) {
		//Toolkit.getDefaultToolkit().beep();
		System.out.println("Aie!!!");  }

	@Override
	public void init(){
		Button btn = new Button("Ding!");
		btn.setStyle("-fx-font: 42 arial; -fx-base: #b6e7c9;");
		// handle the button clicked event
		btn.setOnAction(this);
		root.getChildren().add(btn);
	}

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

Correction de l'exercice « KeyAdapter » du cours d'Arnaud Pêcher en utilisant API JavaFX




```
public class KeyAdapter extends Application implements EventHandler<KeyEvent> {
	Rectangle r;
	Group root;

	public KeyAdapter() {
		r = new Rectangle();
		root = new Group();
		r.setX(50);  r.setY(50);
		r.setWidth(10);  r.setHeight(10);
		r.setFill(Color.BLACK);
		root.getChildren().add(r);
	}

	@Override
	public void start(Stage primaryStage) throws Exception {
		primaryStage.setTitle("KeyAdapter - Johnen");
		Scene scene = new Scene(root,200,100);
		scene.setOnKeyPressed(this);
		primaryStage.setScene(scene);
		primaryStage.show();   }


	@Override
	public void handle(KeyEvent event) {
		System.out.println("Handle keypress "+event.getCode());
		switch (event.getCode()) {
		case UP:    r.setY(r.getY()-10);  break;
		case DOWN:  r.setY(r.getY()+10);  break;
		case LEFT:  r.setX(r.getX()-10);  break;
		case RIGHT:  r.setX(r.getX()+10);  break;
		case P:    r.setY(r.getY()-10);  break;
		case N:    r.setY(r.getY()+10);  break;
		case B:    r.setX(r.getX()-10);  break;
		case F:    r.setX(r.getX()+10);  break;
		default: break;
		}     }

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

Correction de l'exercice « Ellipse » du cours d'Arnaud Pêcher en utilisant JavaFX




```
public class MainEllipse extends Application {
	private Ellipse ellipse;

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

	public void FenetreVide(Stage stage,String titre, int hauteur, int largeur)
	   { stage.setWidth(largeur); stage.setHeight(hauteur);
	     stage.setTitle(titre);  }

	public HBox creerLigneBoutton() {
		HBox hbox = new HBox(10);
		hbox.setStyle("-fx-background-color: BEIGE;");
		hbox.setAlignment(Pos.CENTER);

		Button buttonPlus = new Button("BLUE");
		buttonPlus.setPrefSize(50, 10);
		buttonPlus.setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event) {
				ellipse.setFill(Color.BLUE);	}});

		Button buttonMoins = new Button("RED");
		buttonMoins.setPrefSize(50, 10);
		buttonMoins.setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event) {
				ellipse.setFill(Color.RED);
		}});

		hbox.getChildren().addAll(buttonPlus, buttonMoins);
		return hbox;
	}

	public BorderPane creerContenu (){
		BorderPane  border = new BorderPane();
		border.setTop(creerLigneBoutton());
		ellipse = new Ellipse( 50,50, 10, 40);
		ellipse.setFill(Color.GREY);
		border.setCenter(ellipse);
		return border;
	}
```

```
public void layout(Stage stage) {
             FenetreVide(stage, "Ellipse - Johnen",100,270);
             BorderPane  border = creerContenu();
             Scene scene = new Scene(border);
             stage.setScene(scene);
             stage.show();   }

      @Override
      public void start(Stage stage){ layout(stage);  }

}
```

Plusieurs écouteurs sur le même bouton

Usage de la classe `GridPane` avec deux colonnes de la même taille.
Exemple d'usage de la méthode `setStyle`.
Deux actions sont réalisées lorsque l'on clique sur l'un des boutons ; lequel ?

```
public class MultiEcoute extends Application implements EventHandler<ActionEvent>
{

      private Button aButton, bButton;
      private TextArea righTextArea, leftTextArea;
      private Label rightLabel, leftLabel;
```

```
    public MultiEcoute() {
       aButton = new Button("bonjour Alice"); aButton.setOnAction(this);
       aButton.setStyle("-fx-font: 14 arial; -fx-border-color: burlywood;
                 -fx-border-width: 5px; -fx-text-fill: #0000ff");

       bButton = new Button("bonjour Bob"); bButton.setOnAction(this);
       bButton.setStyle("-fx-font: 14 arial; -fx-border-color: burlywood;
                 -fx-border-width: 5px;-fx-text-fill: #0000ff");

       EventHandler<ActionEvent> handler2 = new EventHandler<ActionEvent>() {
          @Override
          public void handle(ActionEvent event) {
                 String st = ((Button) event.getSource()).getText()+"\n";
                 righTextArea.appendText(st);  }};

       aButton.addEventHandler(ActionEvent.ACTION, handler2);

        righTextArea = new TextArea(); righTextArea.setPrefColumnCount(200);
        righTextArea.setPrefRowCount(10); righTextArea.setEditable(false);

        leftTextArea = new TextArea(); leftTextArea.setEditable(false);

        rightLabel = new Label("listener hears");
        rightLabel.setStyle("-fx-font: 16 arial;");
        leftLabel = new Label("NOSY listener hears");
        leftLabel.setStyle("-fx-font: 16 arial;-fx-text-fill: red;
                 -fx-font-weight: bold;");
    }

    @Override
    public void handle(ActionEvent ev) {
          String st = ((Button) ev.getSource()).getText()+"\n";
          leftTextArea.appendText(st);  }


   public Pane miseEnplace() {
      GridPane pane = new GridPane();
     ColumnConstraints col1 = new ColumnConstraints(); col1.setPercentWidth(50);
     ColumnConstraints col2 = new ColumnConstraints(); col2.setPercentWidth(50);
      pane.getColumnConstraints().addAll(col1,col2);
      pane.setHgap(10);
      pane.add(rightLabel, 0, 0); pane.add(righTextArea, 0,1);
      pane.add(leftLabel, 1, 0); pane.add(leftTextArea, 1,1);
      pane.add(aButton, 0, 2); pane.add(bButton, 1, 2);
      return pane;  }

    @Override
    public void start(Stage stage) throws Exception {
          stage.setWidth(400); stage.setHeight(350);
          stage.setTitle("Multiplexeur Johnen");
          Pane pane = miseEnplace();
          Scene scene = new Scene(pane);
          stage.setScene(scene);
          stage.show();    }

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

}
```

Questions :

1. Ajouter un intervenant supplémentaire « Claude » (une colonne) ; les trois colonnes doivent s'ajuster à la largeur de la fenêtre ;
2. La colonne « Alice » doit être deux fois plus large que la colonne « Bob » et que celle de « Claude » ;
3. Modifier le style des labels (font, couleur, gras, …) ;
4. Modifier la couleur du fond de `rightTextAera.`

Propriétés en JavaFX et le patron de conception « Observable/Observer »

L'API Observer/Observable de Java est incompatible avec JavaFX.
La solution est d'utiliser les « propriétés » avec les interfaces `ObservableValue<T>` et `ChangeListener<T>`.

```
public class Personn {
     private final IntegerProperty age = new SimpleIntegerProperty(0);

      public Personn () { }
      public final int getAge() {  return age.get(); }
      public final void setAge(int value) { age.set(value);  }

       // Accès à la propriété.
      public final IntegerProperty ageProperty() { return age;  }

      public void addListenerAge(ChangeListener<Number> cl) {age.addListener(cl);}
}

public class Main  {
      public static void main(String[] args) {
           Personn prs = new Personn();

           // Adding a change listener with anonymous inner class
            prs.addListenerAge(new ChangeListener<Number>() {
               @Override
                public void changed(ObservableValue<? extends Number> arg0,
                       Number oldVal, Number newVal) {
                   System.out.println("\n old age: "+oldVal);
                   System.out.println(" new age: "+newVal);
               }   });

           // Adding a change listener with anonymous inner class
            prs.addListenerAge(new ChangeListener<Number>() {
                @Override
                public void changed(ObservableValue<? extends Number> arg0,
                          Number oldVal, Number newVal) {
                    if ((Integer)newVal < 18)
                         System.err.println("tu es mineur, tu as "+newVal+" ans");
                    if ((Integer)newVal > 65)
                        System.err.println("tu es vieux, tu as "+newVal+" ans");
                }        });

           prs.setAge(9) ;   prs.setAge(25) ;   prs.setAge(67) ;
      }
}
```

XXXProperty implements ObservableValue<Number>

- javafx.beans.property.BooleanProperty - permet de stocker des valeurs booléennes ;
- javafx.beans.property.IntegerProperty - permet de stocker des valeurs entières ;
- javafx.beans.property.LongProperty - permet de stocker des valeurs entières longues ;
- javafx.beans.property.FloatProperty - permet de stocker des valeurs flottantes ;
- javafx.beans.property.DoubleProperty - permet de stocker des valeurs flottantes en précision double ;
- javafx.beans.property.StringProperty - permet de stocker des chaines de caractères.

Il n'existe pas de stockage dédié pour les « byte », « char » et « short ». Il faudra donc utiliser des IntegerProperty ou un ObjectProperty (voir plus bas) pour stocker de telles valeurs.

Chacun de ces types dispose de classes parentes qui ne permettent que la lecture seule (ex : ReadOnlyIntegerProperty) et de classes concrètes filles prêtes à l'emploi permettant la lecture-écriture (ex : SimpleIntegerProperty).