



COURS JAVA AVANCÉ – PARTIE 3

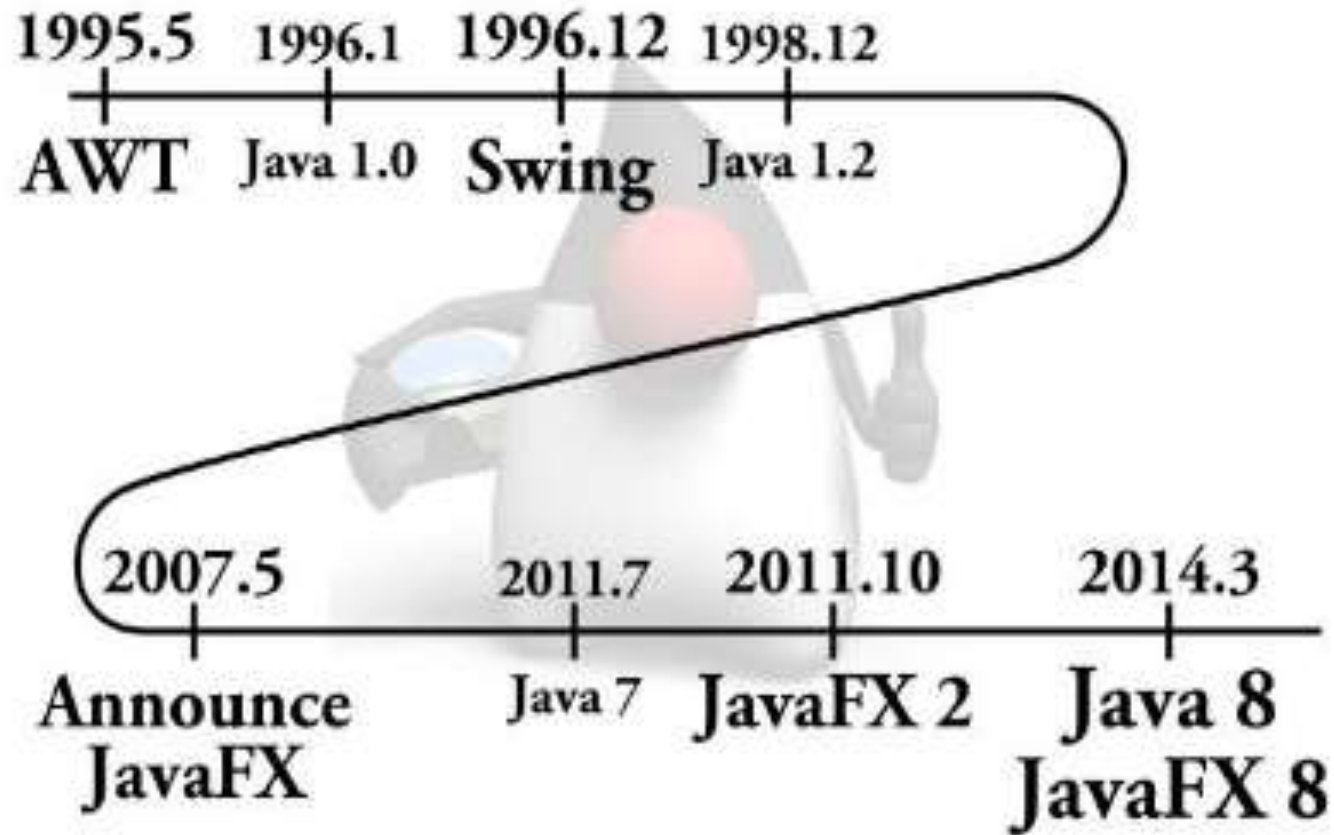
Mme Nouria Sana

A series of five orange circles of varying sizes arranged in a descending, slightly curved pattern on the left side of the slide.

IHM- JAVA FX

Mme Nourira Sana

L'EVOLUTION DANS LE TEMPS



PLATEFORME JAVAFX

- **Manière déclarative** en décrivant l'interface dans un fichier FXML (syntaxe XML). L'utilitaire graphique **Scene Builder** facilite la création et la gestion des fichiers FXML. L'interface peut être créée par un designer (sans connaissance Java, ou presque...). On aura aussi séparation entre présentation et logique de l'application (MVC)
- **Manière procédurale** en utilisant d'**API** pour construire l'interface avec du code Java. Ainsi, la création et manipulation des interfaces seront dynamiques, et la création d'extensions et variantes se fait par héritage.

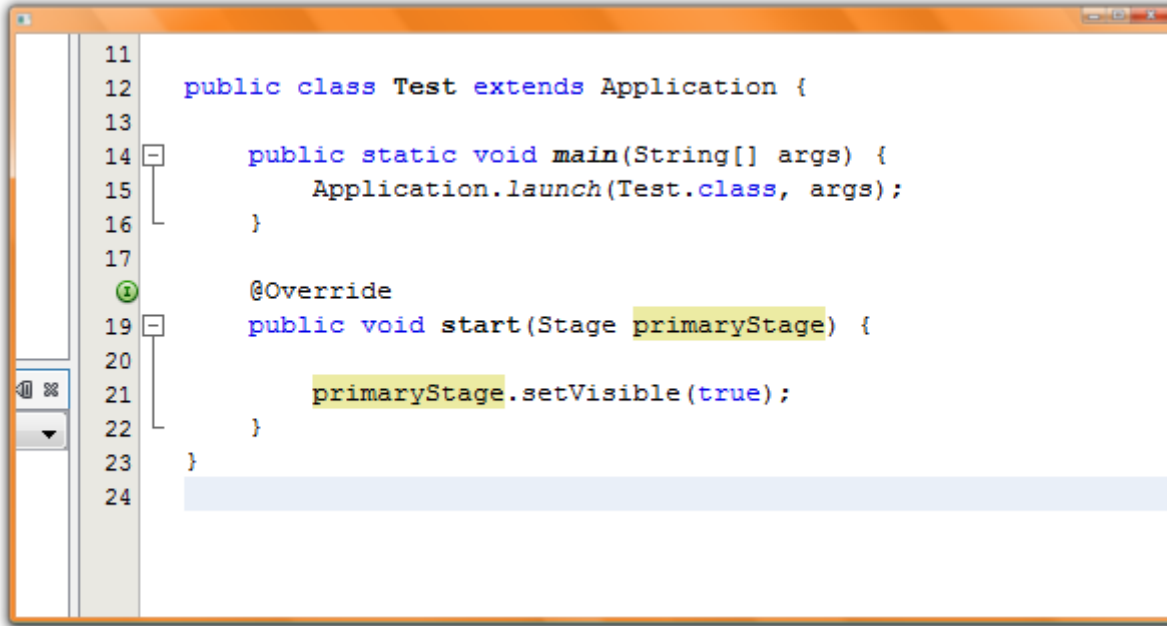


CREATION D'UNE APPLICATION JAVAFX

- créez un nouveau projet en allant dans la barre de menu : File >> New Project...,
- puis dans la rubrique "Categories" sélectionnez Java et dans la rubrique "Projects" sélectionnez Java FX Application.
- Cliquez sur Next, nommez le projet "Test" et cliquez sur Finish.



CLASSE TEST AVEC JAVAFX



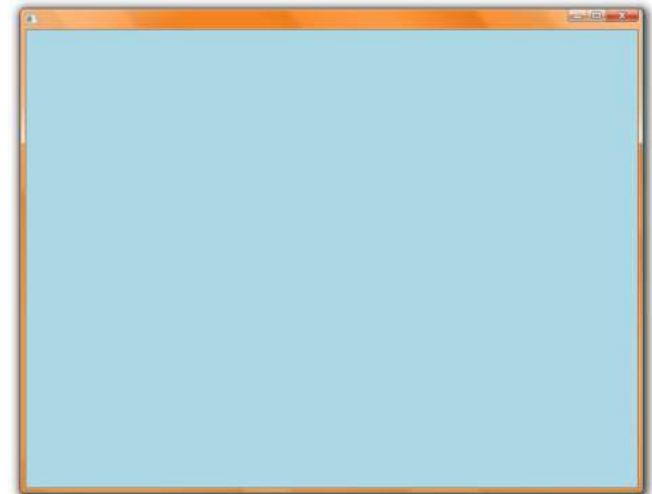
```
11
12 public class Test extends Application {
13
14     public static void main(String[] args) {
15         Application.launch(Test.class, args);
16     }
17
18     @Override
19     public void start(Stage primaryStage) {
20
21         primaryStage.setVisible(true);
22     }
23 }
24
```

METHODE START()

@Override

```
public void start(Stage primaryStage) {
    Group root = new Group();
    Scene scene = new Scene(root, 800, 600,
Color.LIGHTBLUE);
    primaryStage.setScene(scene);

    primaryStage.setVisible(true);
}
```

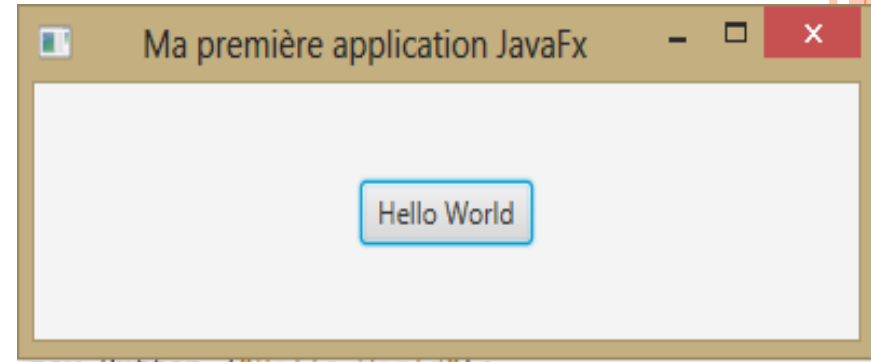


- L'application est codée en créant une sous-classe de **Application**
- La méthode **start()** construit le tout
 - La fonction *start()*. Cette fonction est déclenchée par la fonction *launch()*, elle prend en argument un objet de type Stage (fenetre)
 - **la fonction start() ne fait que rendre visible l'objet Stage, c'est-à-dire la fenêtre de notre application**
- La fenêtre principale d'une application est représentée par un objet de type **Stage** qui est fourni par le système au lancement de l'application.
- L'interface est représentée par un objet de type **Scene** qu'il faut créer et associer à la fenêtre (Stage).
- La scène est composée des différents éléments de l'interface graphique (composants de l'interface graphique) qui sont des objets de type **Node**.
- La fonction *main()*. : le point d'entrée d'une application. Elle appelle la fonction *launch()* qui lancera le reste du programme. C'est la seule instruction que doit contenir la fonction *main()*.

PREMIER PROGRAMME HELLO WORD

- PAR MANIÈRE PROCÉDURALE

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
primaryStage.setTitle("Ma première application JavaFx"); //définit le titre de la fenêtre (affiché selon OS)
        BorderPane root = new BorderPane(); // root : politique de positionnement
        Button bk = new Button ("Hello World"); // composant
        root.setCenter(bk); // composant ajouter au root
Scene scene = new Scene(root, 250,100); // root initialize la scene
primaryStage.setScene(scene); // définit la scène (sa racine) qui est associée à la fenêtre
primaryStage.show(); //affiche la fenêtre à l'écran (et la scène qu'elle contient)
    }
    public static void main(String[] args) {
        launch(args); // main contient une seule méthode launch
    }
}
```



EXAMPLE 2 :

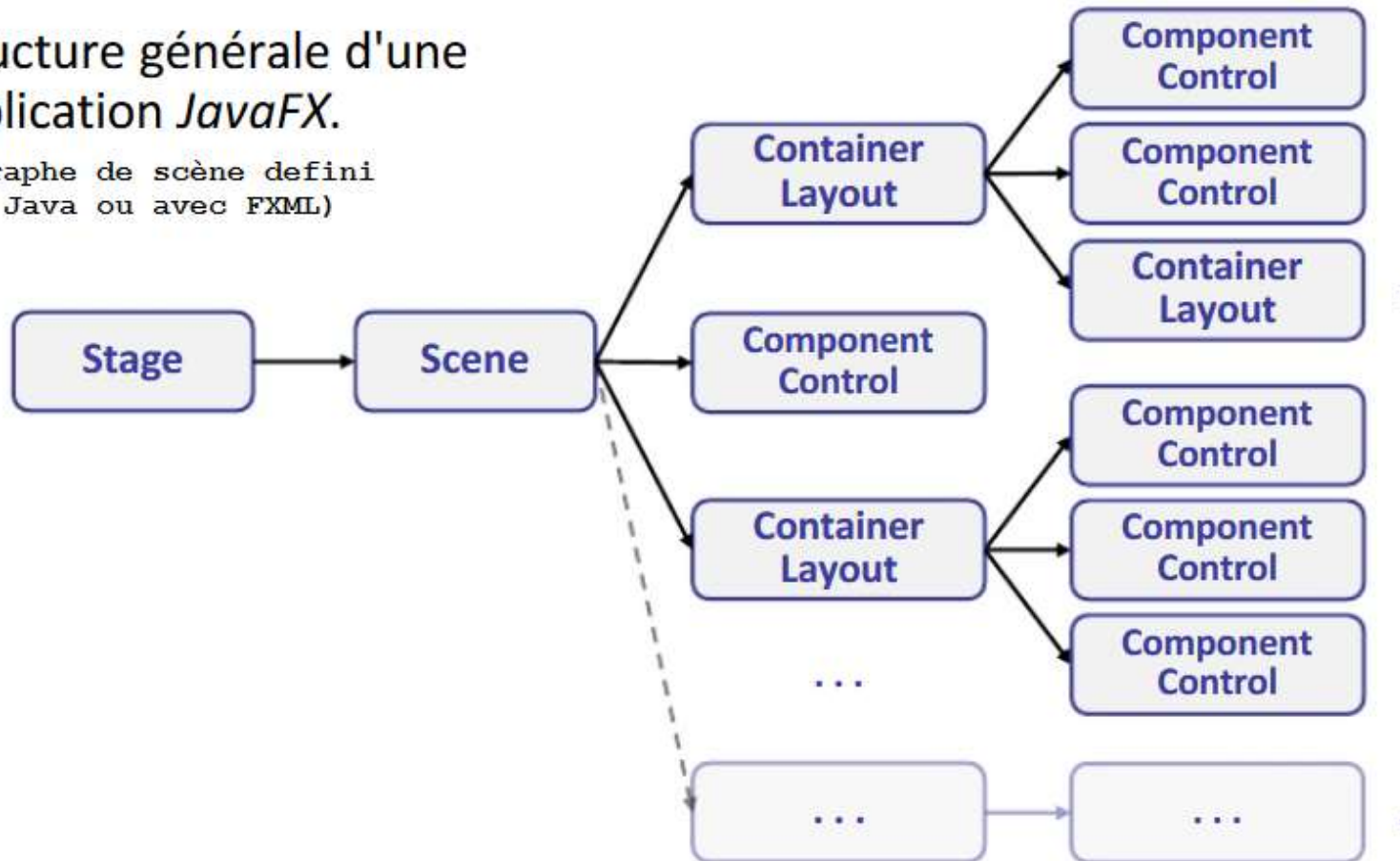
```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class SayHello extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Ma première application JavaFx");
        BorderPane root = new BorderPane();
        Button bk = new Button();
        bk.setText("Say Hello");
        root.setCenter(bk);
        bk.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!"); }}
        Scene scene = new Scene(root, 250,100);
        primaryStage.setScene(scene);
        primaryStage.show(); }
    public static void main(String[] args) {
        launch(args);  } }
```



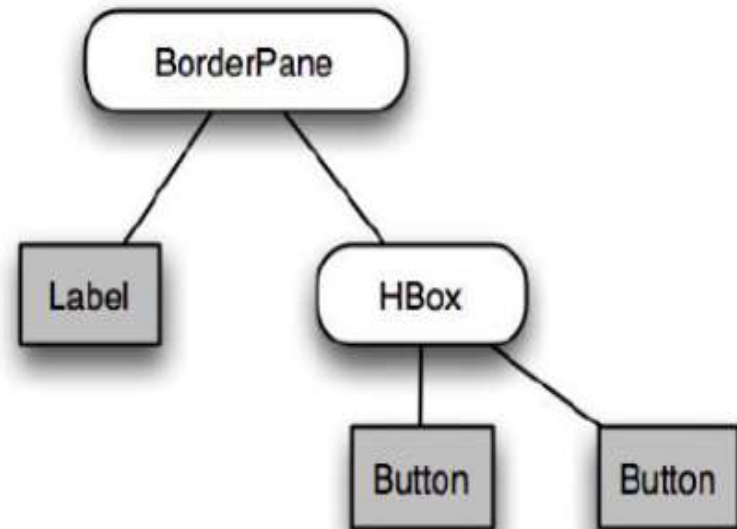
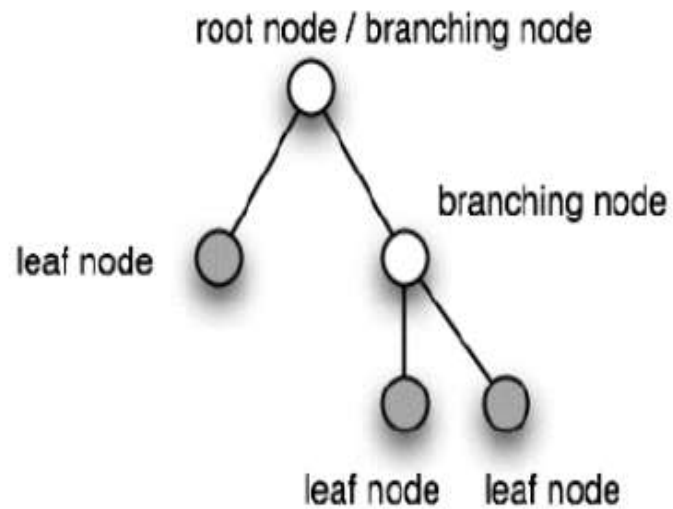
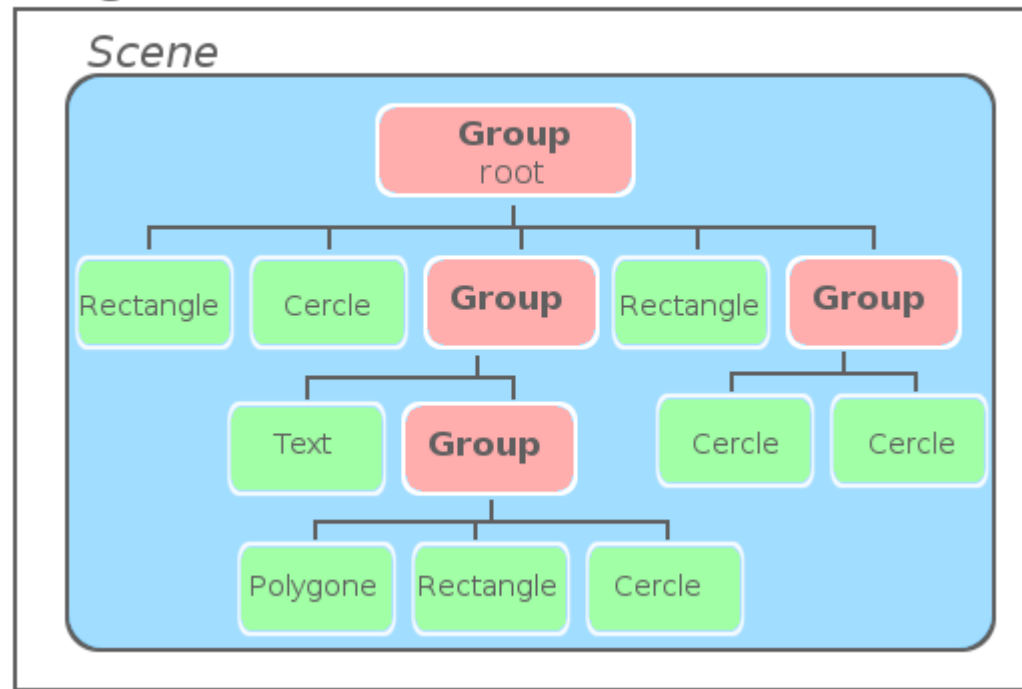
ASPECT ARCHITECTURAL

Structure générale d'une application *JavaFX*.

(Graphe de scène défini en Java ou avec FXML)



Stage



CONCEPTS TECHNIQUES

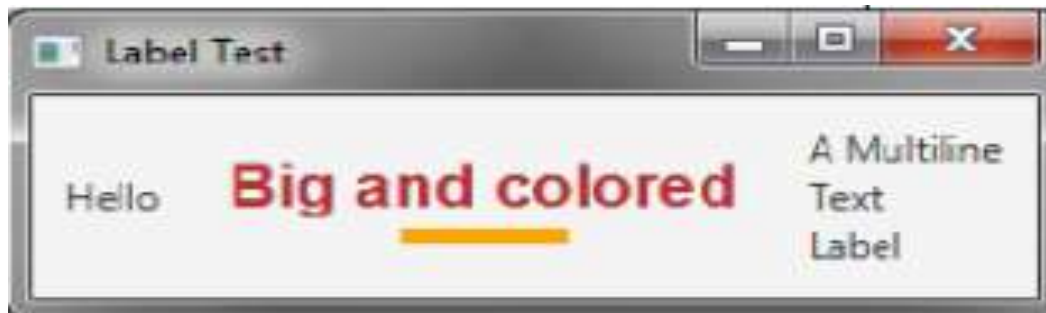
- On voit que l'objet Scene contient un groupe *root*. C'est le groupe racine qui contiendra tous les autres objets et groupes d'objets graphiques. Un objet Scene ne contient qu'un seul groupe *root*.
- A partir de ce groupe root, on peut insérer et retirer tous les nœuds graphiques que l'on souhaite.
- Un nœud graphique peut être :
 - Un objet graphique comme un cercle, un rectangle, une image, etc.
 - Un groupe contenant des objets graphiques, un groupe peut même contenir d'autres groupes.
 - Ou un type d'objet graphique que nous avons nous-même défini



LES COMPOSANTS

Label représente un libellé (= un texte non éditable)

```
//constructeurs
private Label lblA = new Label("Hello");
private Label lblB = new Label("Big and colored");
private Label lblC = new Label("A Multiline\nText\nLabel");
// quelques méthodes
lblB.setFont(Font.font("SansSerif", FontWeight.BOLD, 20));
lblB.setTextFill(Color.rgb(180, 50, 50));
lblB.setGraphic(new Rectangle(50, 5, Color.ORANGE));
lblB.setContentDisplay(ContentDisplay.BOTTOM);
```



BUTTON

```
private Button btnOk = new Button("OK");  
private Button btnLogin = new Button("Login");  
private Button btnSave = new Button("Save");  
private Button btnMulti = new Button("A Multiline\nRight-Justified\nText");  
private static final String FLOGO = "/resources/EIA_FR.jpg";  
// quelques méthodes  
btnLogin.setTextFill(Color.BLUE);  
btnLogin.setFont(Font.font(null, FontWeight.BOLD, 14));  
btnLogin.setGraphic(new ImageView(FLOGO));  
btnLogin.setContentDisplay(ContentDisplay.TOP);  
btnSave.setDefaultButton(true);  
btnMulti.setTextAlignment(TextAlignment.RIGHT);
```

Button

représente un bouton permettant à l'utilisateur de déclencher une action.



ToggleButton (bouton) représente un bouton bistable. Il comporte donc deux états : un clic le met à l'état sélectionné (on), un nouveau clic le remet à l'état désélectionné (off)

```
private ToggleButton btnGif = new ToggleButton("GIF");  
private ToggleButton btnJpg = new ToggleButton("JPEG");  
    private ToggleButton btnPng = new ToggleButton("PNG");  
    // quelques methods  
    ToggleGroup tg = new ToggleGroup();  
btnGif.setToggleGroup(tg); btnJpg.setToggleGroup(tg);  
    btnPng.setToggleGroup (tg);
```



TEXTFIELD

```
private Label lblName = new Label("Name");  
private Label lblMobile = new Label("Mobile");  
private TextField tfdName = new TextField();  
    private TextField tfdMobile = new TextField();  
    // quelques méthodes  
tfdName.setPrefColumnCount(12);  
tfdName.setPromptText("First and Last-Name");  
tfdMobile.setPrefColumnCount(8);  
tfdMobile.setPromptText("Mobile Tel Nr");
```

TextField représente un champ texte d'une seule ligne qui est éditable par défaut mais qui peut également être utilisé pour afficher du texte. Il n'est pas libellé.

PasswordField est **une sous-classe TextField** Et permet de saisir un texte sans que celui-ci soit affiché dans le champ texte (il est remplacé par des caractères de substitution). Elle utilise les memes méthodes que TextField.



Il existe Sous classe PasswordField : saisir un texte sans que celui-ci soit affiché

TEXTAREA

```
private TextArea txaA = new TextArea();  
private Button btnB = new Button("Print");  
// Quelques methods  
txaA.setWrapText(true);  
txaA.setPrefColumnCount(14);  
txaA.setPrefRowCount(8);
```

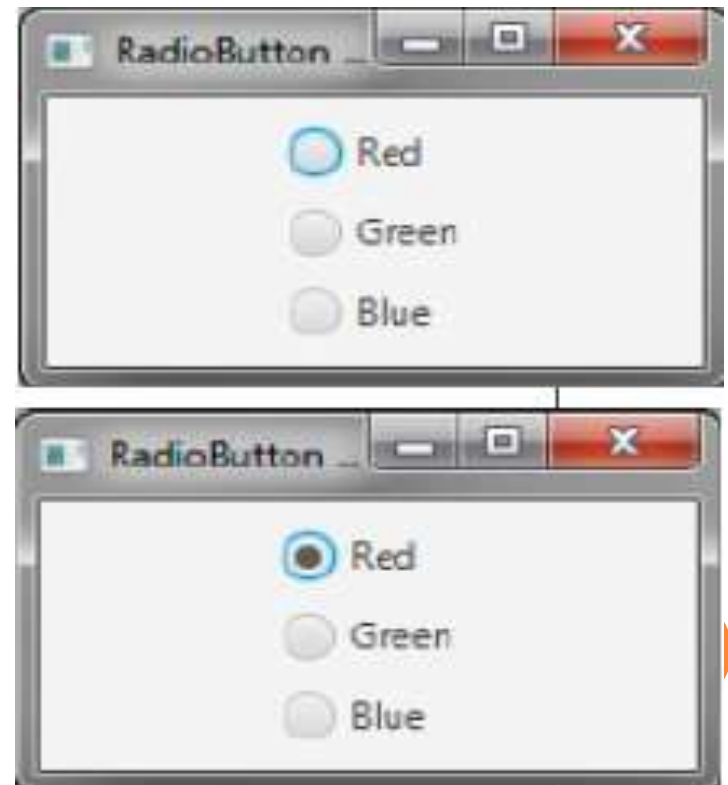
TextArea permet d'afficher et de saisir du texte dans un champ multiligne (une zone de texte).
Le texte peut être renvoyé à la ligne automatiquement (wrapping) et des barres de défilement (scrollbar) horizontales et/ou verticales sont ajoutées automatiquement si la taille du composant ne permet pas d'afficher l'entier du texte (lignes ou colonnes tronquées).



```
private RadioButton rbnGif = new RadioButton("Red");  
private RadioButton rbnJpg = new RadioButton("Green");  
private RadioButton rbnPng = new RadioButton("Blue");  
// quelques méthodes  
// les boutons radio font partie d'une groupe  
    ToggleGroup tg = new ToggleGroup();
```

```
rbnGif.setToggleGroup(tg);  
rbnJpg.setToggleGroup(tg);  
rbnPng.setToggleGroup(tg);
```

RadioButton est une sous-classe de **ToggleButton** et représente une option que l'utilisateur peut sélectionner (généralement parmi un groupe d'options).
L'utilisation du composant **RadioButton** doit être réservée à la situation où l'utilisateur doit choisir une seule option parmi plusieurs (même si le composant n'impose pas cette sémantique).

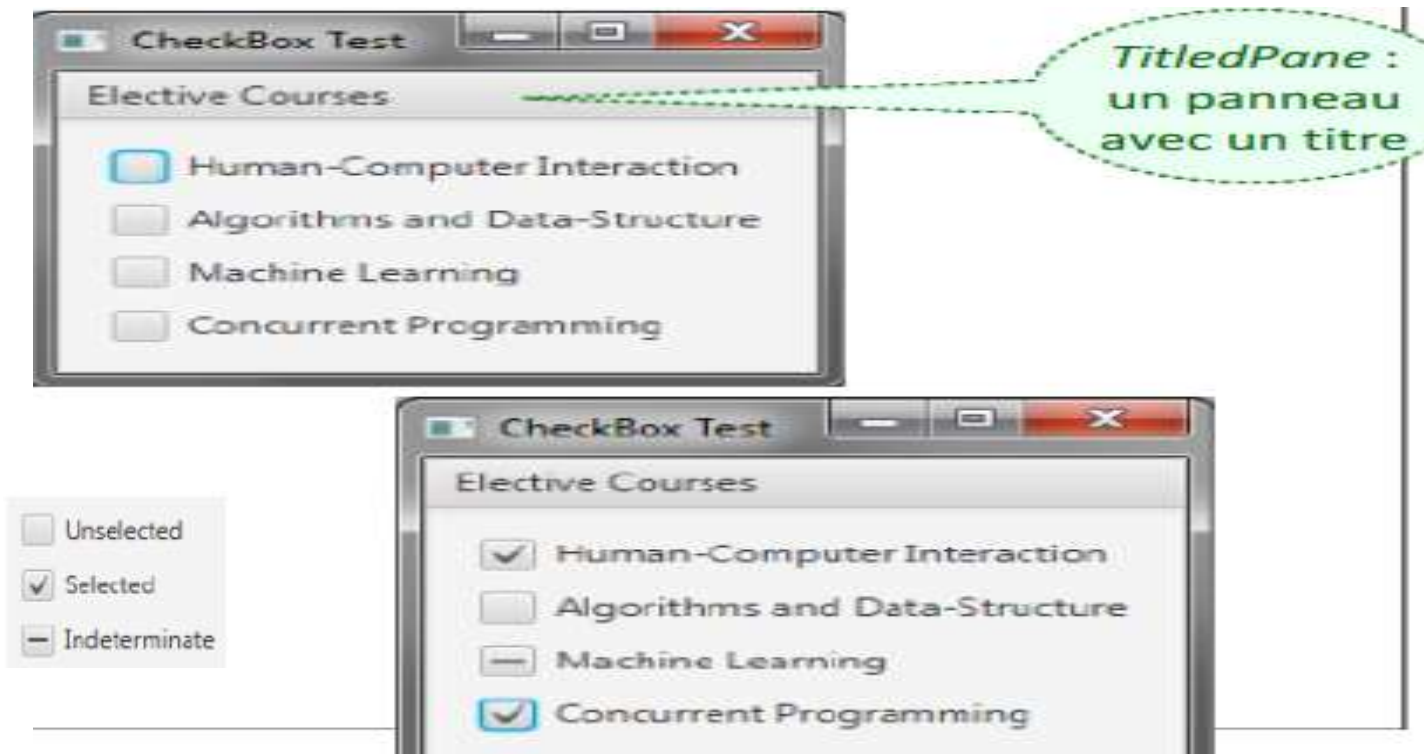


```
private CheckBox chbHci = new CheckBox("Human-Computer Interaction");  
private CheckBox chbAlgo = new CheckBox("Algorithms and Data-Structure");  
private CheckBox chbMl = new CheckBox("Machine Learning");  
private CheckBox chbCp = new CheckBox("Concurrent Programming")
```

// quelques methods

```
chbHci.setAllowIndeterminate(true);  
chbAlgo.setAllowIndeterminate(true);  
chbMl.setAllowIndeterminate(true);  
chbCp.setAllowIndeterminate(true);
```

CheckBox représente une case à cocher que l'utilisateur peut sélectionner ou désélectionner parmi plusieurs options qui peuvent être simultanément

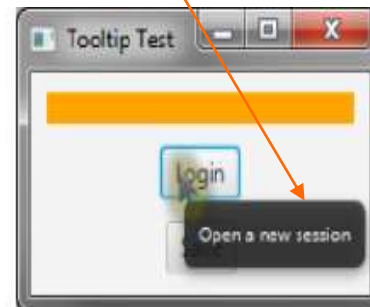
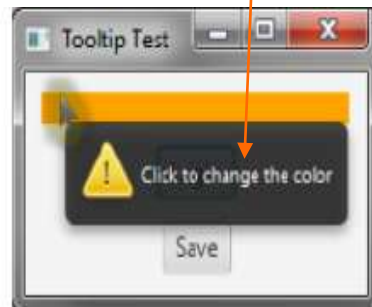
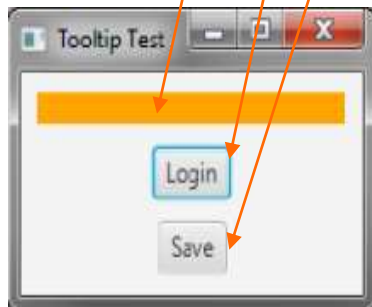


```
private Rectangle rect = new Rectangle(180, 15, Color.ORANGE);  
private Button btnLogin = new Button("Login");  
private Button btnSave = new Button("Save");
```

```
private Tooltip ttpRect = new Tooltip("Click to change the color");  
private Tooltip ttpLogin = new Tooltip("Open a new session");
```

```
// quelques méthodes  
ttpRect.setGraphic(new ImageView(ICON));  
Tooltip.install(rect,ttpRect);  
btnLogin.setTooltip(tpLogin);
```

Tooltip permet d'afficher une bulle d'aide ou bulle d'information lorsque l'utilisateur s'arrête sur un des éléments de l'interface avec le curseur de la souris (ou autre dispositif de pointage)



- **Hyperlink** se présente comme un lien hypertexte HTML dans une page web. Il agit comme un bouton lorsqu'on clique sur le texte associé et peut déclencher n'importe quelle action (comme un bouton).
- Ce composant est une sous-classe de **ButtonBase** et de **Labeled**. Il peut donc contenir un autre composant (**graphic**), typiquement une image ou un graphique.
- Les constructeurs permettent de définir le contenu du texte et d'un éventuel composant additionnel (**graphic**) :
 - `new Hyperlink("www.myblog.ch");`
 - `new Hyperlink("Print", printerIcon);`
- Le composant **Hyperlink** ne comporte qu'une propriété spécifique :
 - `visited`. Elle indique si le lien a déjà été visité, c'est-à-dire cliqué (**Boolean**).
- Si l'on souhaite que le lien hypertexte affiche le contenu d'une page web, on peut utiliser le composant **WebView** qui contient un moteur de rendu de pages web de type **WebEngine** (basé sur le projet open-source **WebKit**).

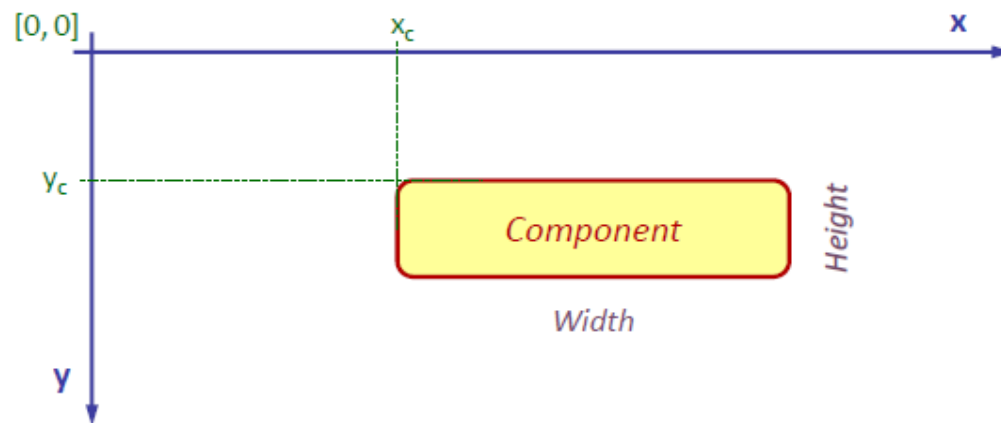


LES CONTENEURS

- Gestionnaires de disposition (layoutmanagers)
- Avec JavaFX, les layoutmanagers sont intégrés aux conteneurs (**layout-panes**)
- Quand on parle de la disposition (layout) d'une interface, on s'intéresse plus particulièrement à :
 - la taille des composants
 - la position des composants
 - la position dans la fenêtre
 - la position relative des éléments les uns par rapport aux autres
 - les alignements et espacements qui favorisent la structuration visuelle
 - les bordures et les marges (notamment autour des conteneurs)
 - le comportement dynamique de l'interface lorsqu'on redimensionne la fenêtre



SYSTÈME DE COORDONNÉES



CONTENEUR- LAYOUT

- Layout : C'est une structure graphique dans laquelle vous pouvez ranger toutes sortes d'objets graphiques de façon bien organisée. **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.



- **HBox** place les composants sur une ligne horizontale. Les composants sont ajoutés à la suite les uns des autres (de gauche à droite).
- La propriété padding permet de définir l'**espace (marge)** entre le bord du conteneur et les composants enfants. Un paramètre de type Insets est passé en paramètre, il définit les espacements dans l'ordre suivant : *Top, Right, Bottom, Left* ou une valeur identique pour les quatre côtés si l'on passe un seul paramètre.
- Une **couleur de fond** peut être appliquée au conteneur en définissant la propriété style qui permet de passer en chaîne de caractères, un style CSS.



```
private Hbox root;  
private Button btnA= new Button("Alpha");  
private Label lblB = new Label("Bravo");  
// constructeur du composant ComboBox  
private ComboBox<String> cbbC= new ComboBox<>();  
  
// quelques méthodes  
root= new HBox(10); // Horizontal Spacing : 10  
root.setAlignment(Pos.CENTER);  
root.setPadding(new Insets(20, 10, 20, 10));  
root.setStyle("-fx-background-color: #FFFE99");  
root.getChildren().add(btnA); // ajout du premier bouton  
root.getChildren().add(lblB);  
cbbC.getItems().addAll("Charlie", "Delta");  
cbbC.getSelectionModel().select(0);  
root.getChildren().add(cbbC);
```



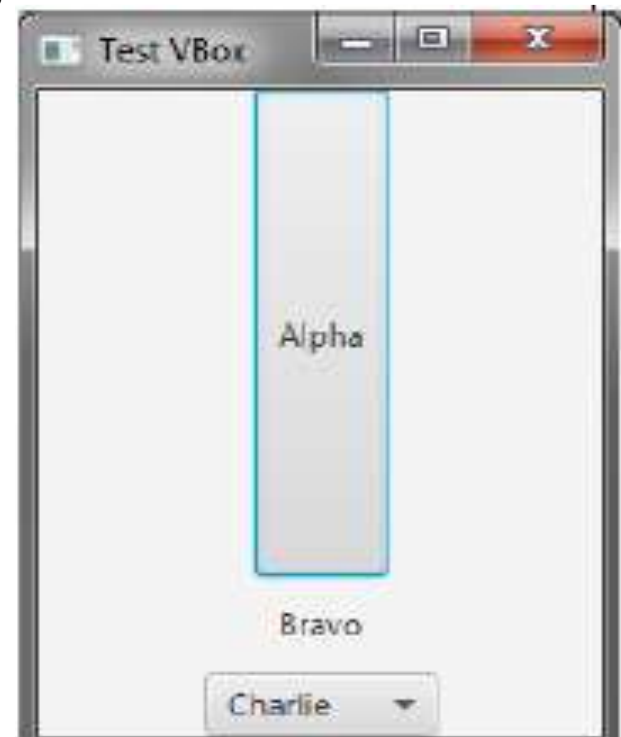
groupe.getChildren().add(objet), permet d'ajouter un nœud graphique à un groupe quelconque.

- **VBox** place les composants verticalement, sur une colonne. Les composants sont ainsi ajoutés à la suite les uns des autres (de haut en bas).
- Des méthodes statiques de VBox peuvent être invoquées pour appliquer des contraintes de positionnement :
 - Vgrow() (Hgrow pour HBox): permet d'agrandir le composant passé en paramètre jusqu'à sa taille maximale selon la priorité (Priority) donnée margin(): fixe une marge (objet de type Insets) autour du composant passé en paramètre (zéro par défaut Insets.EMPTY)



```
// constructeurs  
private VBox root;  
private Button btnA= new Button("Alpha");  
private Label lblB= new Label("Bravo");  
private ComboBox<String> cbbC= new ComboBox<>();
```

```
// quelques méthodes  
VBox.setVgrow(btnA, Priority.ALWAYS);  
btnA.setMaxHeight(Double.MAX_VALUE);  
root= new VBox(10); // Vertical Spacing : 10  
root.setAlignment(Pos.TOP_CENTER);  
root.getChildren().add(btnA);  
root.getChildren().add(lblB);  
cbbC.getItems().addAll("Charlie", "Delta");  
cbbC.getSelectionModel().select(0);  
root.getChildren().add(cbbC);
```



- **FlowPane** place les composants sur une ligne horizontale ou verticale et passe à la ligne ou à la colonne suivante (*wrapping*) lorsqu'il n'y a plus assez de place disponible.
- Un des paramètres du constructeur (de type *Orientation*) détermine s'il s'agit d'un FlowPane horizontal (par défaut) ou vertical.
- L'ajout des composants enfants dans un conteneur FlowPane s'effectue en invoquant `getChildren().add(node)` ou `addAll(n, ...)`



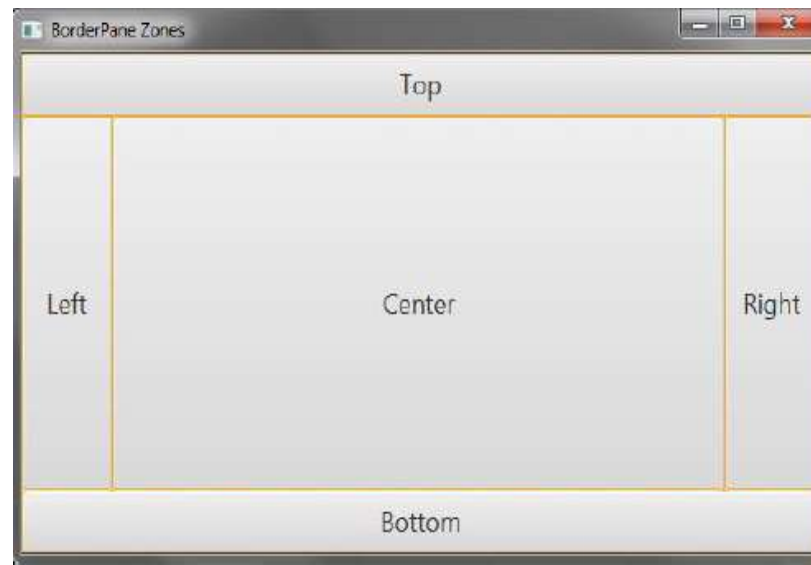
FLOWPANE

.....

```
root= newFlowPane();  
root.getChildren().add(btnA);// Ajout  
root.getChildren().add(lblB);// des  
root.getChildren().add(btnC);// composants  
root.getChildren().add(lblD);  
root.getChildren().add(btnE);  
root.getChildren().add(lblF);  
root.getChildren().add(btnG);  
root.setPadding(new Insets(5));// Marge extérieure  
root.setHgap(10); // Espacement horiz. entre composants  
root.setVgap(15); // Espacement vertical entre lignes  
root.setPrefWrapLength(250);// Largeur préférée du conteneur  
    root.setRowValignment(VPos.BOTTOM);  
    // Aligement vertical dans lignes
```



- **BorderPane** permet de placer les composants enfants dans cinq zones : *Top*, *Bottom*, *Left*, *Right* et *Center*.
- Un seul objet Node (composant, conteneur, ...) peut être placé dans chacun de ces emplacements.
- Pour placer plusieurs composants dans les zones du BorderPane, il faut y ajouter des noeuds de type conteneur et ajouter ensuite les composants dans ces conteneurs imbriqués.

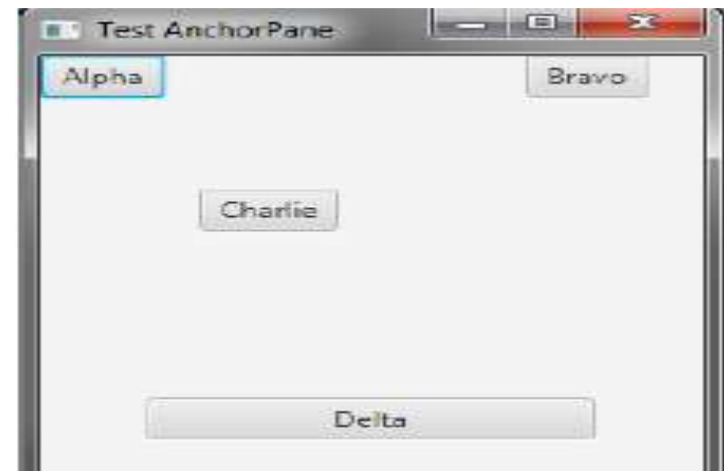
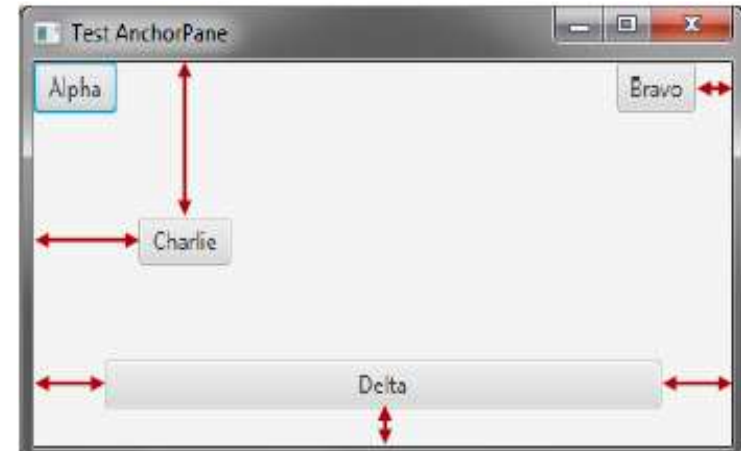


```
private BorderPane root= new BorderPane();  
private HBox btnPanel= new HBox(10);  
private Label lblTitle= new Label ("App Title");  
private Button btnSave= new Button("Save");  
private Button btnQuit= new Button("Quit");  
    private Button btnCancel= new Button("Cancel");  
    // quelques méthodes  
lblTitle.setFont(Font.font("SansSerif", FontWeight.BOLD, 24));  
lblTitle.setTextFill(Color.BLUE);  
root.setTop(lblTitle);  
  
btnPanel.getChildren().add(btnSave);  
btnPanel.getChildren().add(btnQuit);  
btnPanel.getChildren().add(btnCancel);  
btnPanel.setAlignment(Pos.CENTER);  
  
root.setBottom(btnPanel);  
BorderPane.setMargin(lblTitle, new Insets(10, 0, 10, 0));  
BorderPane.setMargin(btnPanel, new Insets(10, 0, 10, 0));  
BorderPane.setAlignment(lblTitle, Pos.CENTER);
```



- **AnchorPane** permet de positionner (ancrer) les composants enfants à une certaine distance des côtés du conteneur (*Top, Bottom, Left et Right*). Il n'est pas divisé en zones.

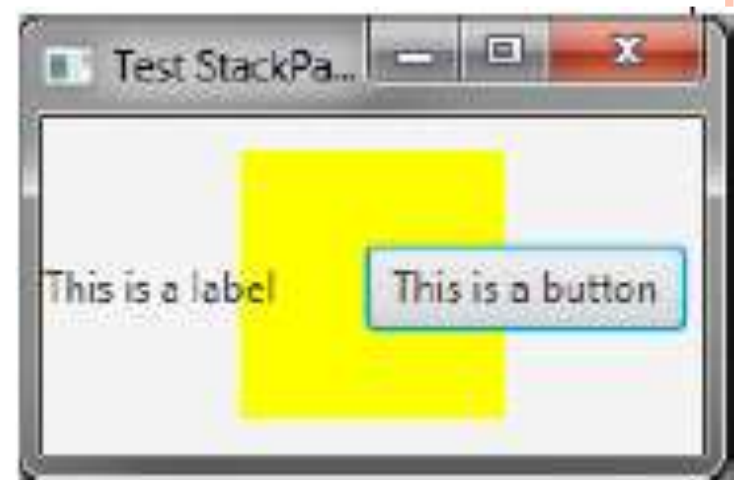
```
private AnchorPane root = new AnchorPane();  
private Button btnA = new Button("Alpha");  
private Button btnB = new Button("Bravo");  
private Button btnC = new Button("Charlie");  
private Button btnD = new Button("Delta");  
    // quelques méthodes  
root.getChildren().addAll(btnA, btnB, btnC, btnD);  
AnchorPane.setRightAnchor(btnB, 20.0);  
AnchorPane.setTopAnchor(btnC, 80.0);  
AnchorPane.setLeftAnchor(btnC, 60.0);  
AnchorPane.setBottomAnchor(btnD, 20.0);  
AnchorPane.setLeftAnchor(btnD, 40.0);  
AnchorPane.setRightAnchor(btnD, 40.0);
```



StackPane empile les composants enfants les uns au dessus des autres dans l'ordre d'insertion : les premiers "au fond", les derniers "au-dessus" (*back-to-front*).

```
private StackPane root= new StackPane();  
private Label lblA= new Label("This is a label");  
private Button btnS= new Button("This is a button");  
    privateRectangle rect= new Rectangle(80,  
    80, Color.YELLOW);
```

```
StackPane.setAlignment(lblA, Pos.CENTER_LEFT);  
StackPane.setAlignment(btnS,  
Pos.CENTER_RIGHT);  
StackPane.setMargin(btnS, new Insets(5));  
root.getChildren().addAll(rect, lblA, btnS);
```



- **GridPane** permet de disposer les composants enfants dans une grille flexible (arrangement en lignes et en colonnes)
- Les **contraintes globales** de lignes/colonnes sont définies dans des objets de type :
 - `RowConstraints`: Pour les lignes
 - `ColumnConstraints`: Pour les colonnes
- en les ajoutant dans une liste, avec les méthodes :
 - `getRowConstraints.add(.....)`
 - `getColumnConstraints.add(.....)`
- L'ordre des ajouts correspond à l'ordre des lignes/colonnes



```
private GridPane root= newGridPane();
private HBox btnPanel= newHBox(12);
private Label lblTitle= newLabel("JafaFXCourse Login");
private Label lblUsername= newLabel("Username or email:");
private TextField tfdUsername= newTextField();
private Label lblPassword= newLabel("Password:");
private Button btnLogin= newButton("Login");
private Button btnCancel= newButton("Cancel");
    private PasswordField pwfPassword= newPasswordField();
    // quelques méthodes
//---Title
lblTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
lblTitle.setTextFill(Color.rgb(80, 80, 180));
root.add(lblTitle, 0, 0, 2, 1);
GridPane.setHalignment(lblTitle, HPos.CENTER);
    GridPane.setMargin(lblTitle, newInsets(0, 0, 10,0));
//---Username(label and text-field)
tfdUsername.setPrefColumnCount(20);
root.add(lblUsername, 0, 1);
root.add(tfdUsername, 1, 1);
GridPane.setHalignment(lblUsername, HPos.RIGHT);
//---Password(label and text-field)
pwfPassword.setPrefColumnCount(12);
```



```

root.add(lblPassword, 0, 2);
root.add(pwfPassword, 1, 2);
GridPane.setHalignment(lblPassword, HPos.RIGHT);
GridPane.setFillWidth(pwfPassword, false);
//---Buttonpanel
btnPanel.getChildren().add(btnLogin);
btnPanel.getChildren().add(btnCancel);
btnPanel.setAlignment(Pos.CENTER_RIGHT);
root.add(btnPanel, 1, 3);
    GridPane.setMargin(btnPanel, new Insets(10, 0, 0,0));
//---Columnglobal constraints
ColumnConstraintsctCol0= newColumnConstraints();
// No constraint
ColumnConstraintsctCol1= newColumnConstraints(50,
200, 400,Priority.ALWAYS, HPos.LEFT,true);
root.getColumnConstraints().add(ctCol0);
root.getColumnConstraints().add(ctCol1);
//---GridPaneproperties
root.setAlignment(Pos.CENTER);
root.setPadding(new Insets(20));
root.setHgap(10);
    root.setVgap(15);

```



;

setGridLinesVisible(true)



- Par défaut, au lancement d'une application, la fenêtre principale (*primarystage*) est centrée sur l'écran.
- Différentes méthodes peuvent être invoquées sur l'objet *Stage* pour influencer la position et la taille de cette fenêtre :
 - *setX()*: position en x du coin supérieur gauche
 - *setY()*: position en y du coin supérieur gauche
 - *centerOnScreen()*: centrage sur l'écran (par défaut)
 - *setMinWidth()*: fixe la largeur minimale de la fenêtre
 - *setMinHeight()*: fixe la hauteur minimale de la fenêtre
 - *setMaxWidth()*: fixe la largeur maximale de la fenêtre
 - *setMaxHeight()*: fixe la hauteur maximale de la fenêtre
 - *setResizable()*: détermine si la fenêtre est redimensionnable
 - *sizeToScene()*: adapte la taille de la fenêtre à la taille de la scène liée à cette fenêtre
 - *setTitle()*: définit le titre de la fenêtre (affiché selon OS)
 - *setFullScreen()*: place la fenêtre en mode plein-écran ou en mode standard (si paramètre *false*) (selon OS)
 - *getIcons().add()*: définit l'icône dans la barre de titre
 - *setAlwaysOnTop()*: place la fenêtre toujours au dessus des autres (généralement à éviter)
 - *setScene()*: définit la scène (sa racine) qui est associée à la fenêtre
 - *show()*: affiche la fenêtre à l'écran (et la scène qu'elle contient)
 - *showAndWait()*: affiche la fenêtre à l'écran et attend que la fenêtre soit fermée pour retourner (méthode bloquante). Cette méthode n'est pas applicable à la fenêtre principale (*primarystage*).

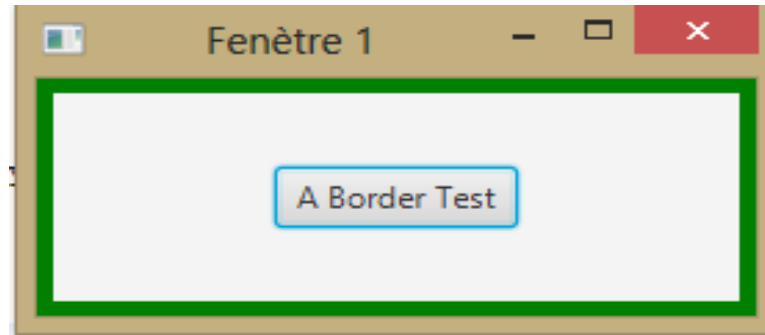


- Pour déterminer la taille de l'écran, on peut utiliser la classe `Screen` et rechercher le rectangle qui englobe la zone utilisable de l'écran (ou l'intégralité de la surface de l'écran)
 - `Screen screen= Screen.getPrimary();`
 - `Rectangle2D bounds= screen.getVisualBounds();`
 - `double screenWidth= bounds.getWidth();`
 - `double screenHeight= bounds.getHeight();`



EXEMPLE BORDURE

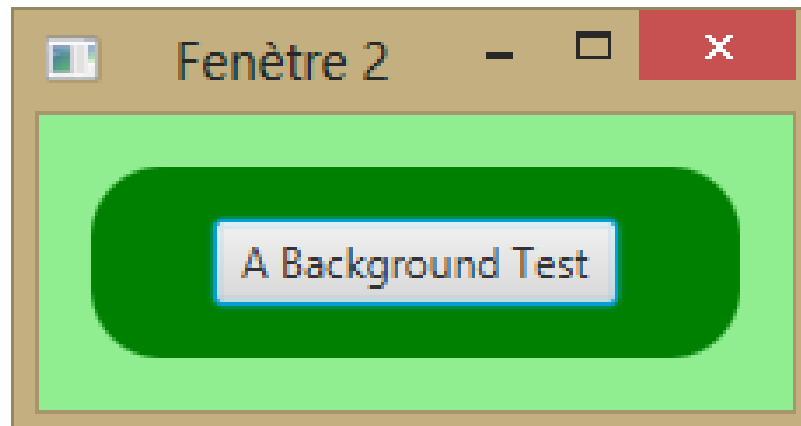
BORDER 1:



BORDER 2



EXEMPLE ARRIERE PLAN



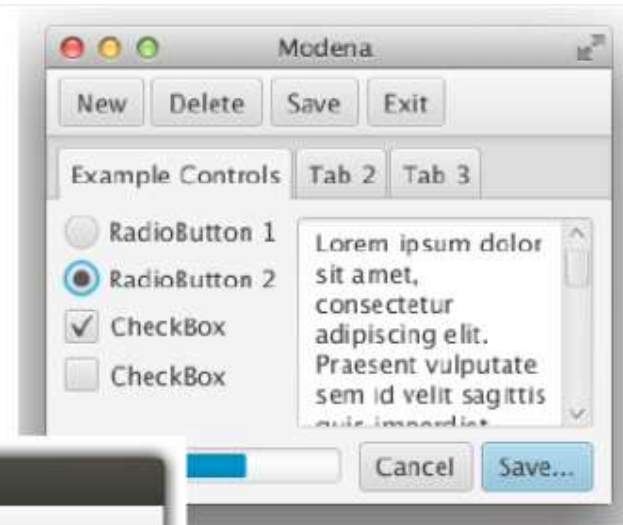
EXEMPLE : CHANGEMENT DE STYLE

@OVERRIDE

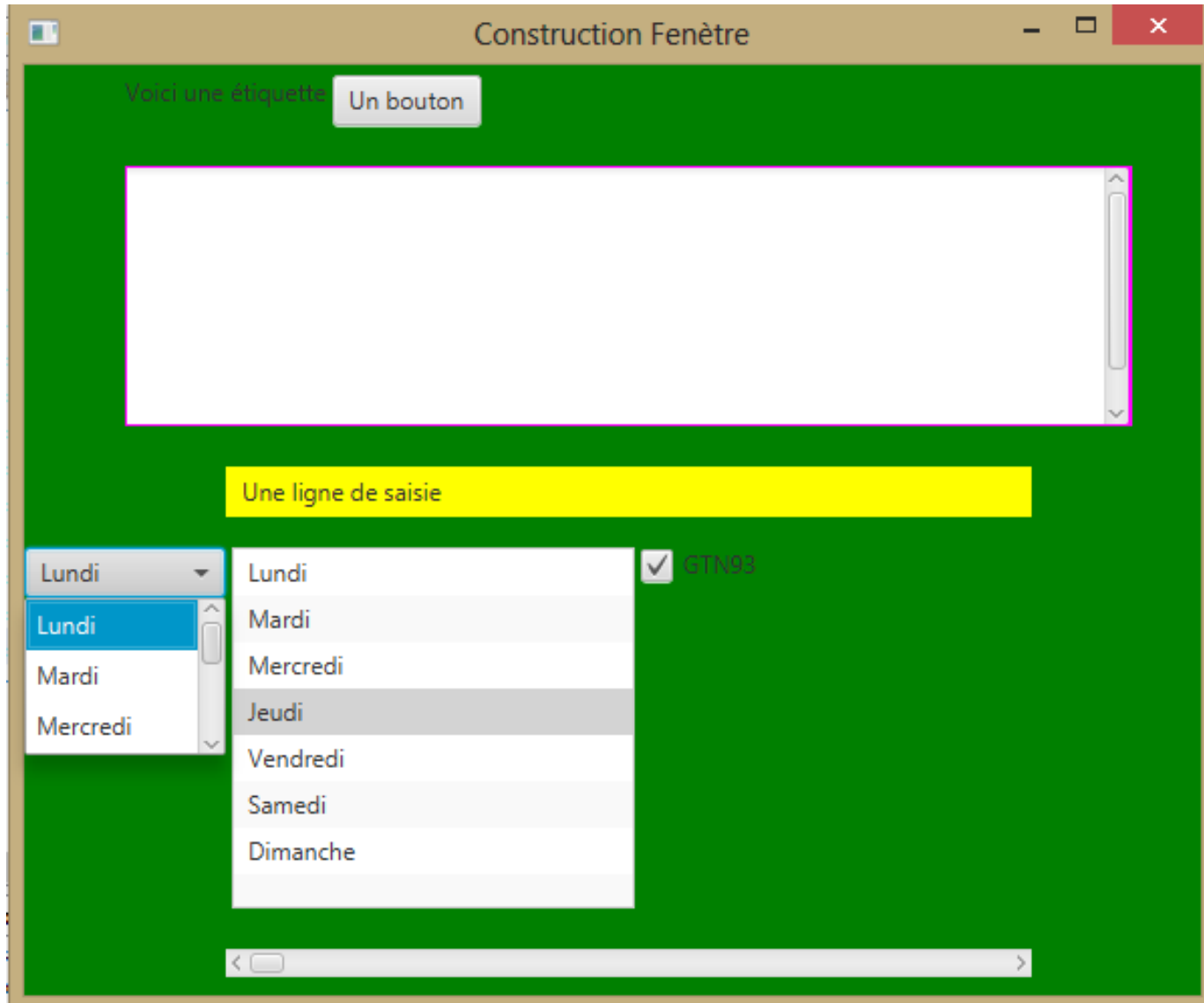
```
PUBLIC VOID START(STAGE PRIMARYSTAGE) { . . .  
    SETUSERAGENTSTYLESHEET ( STYLESHEET_CASPIAN);  
    . . . }
```

REMARQUE : D'AUTRES STYLES EXTERNES PEUVENT ÊTRE IMPORTÉS ET APPLIQUÉS. ON TROUVE DIFFÉRENTES RÉALISATIONS, PAR EXEMPLE : APPLE AQUA (AQUAFX), MICROSOFT MODERN UI (JMETRO), WINDOWS-7 AERO (AEROFX), TWITTER BOOTSTRAP (FEXTILE), FLATTER (EMBEDDED UI) . . .

Style *Modena* sur différents OS.



EXEMPLE : CONSTRUCTION D'UNE INTERFACE



PROGRAMMATION ÉVÈNEMENTIELLE

- Un événement (*event*) constitue une notification qui signale que quelque chose s'est passé. Il peut être provoqué par :
 - une action de l'utilisateur
 - Exemple : un clic avec la souris, la pression sur une touche du clavier, le déplacement d'une fenêtre, un geste sur un écran tactile...
 - un changement provoqué par le système (une valeur a changé (propriété),
 - Exemple : un *timer* est arrivé à échéance, un processus a terminé un calcul, une information est arrivée par le réseau...
- Avec *JavaFX* les événements sont représentés par des objets de la classe **Event** ou, plus généralement, d'une de ses sous-classes.
 - De nombreux événements sont prédéfinis (MouseEvent, KeyEvent, DragEvent, ScrollEvent...) mais il est également possible de créer ses propres événements en créant des sous-classes de Event



Action de l'utilisateur	Événement	Dans class
Pression sur une touche du clavier	KeyEvent	Node, Scene
Déplacement de la souris ou pression sur une de ses touches	MouseEvent	Node, Scene
Glisser-déposer avec la souris (<i>Drag-and-Drop</i>)	MouseDragEvent	Node, Scene
Glisser- déposer propre à la plateforme (geste par exemple)	DragEvent	Node, Scene
Composant "scrollé"	ScrollEvent	Node, Scene
Geste de rotation	RotateEvent	Node, Scene
Geste de balayage/ défilement (Swipe)	SwipeEvent	Node, Scene
Un composant est touché	TouchEvent	Node, Scene
Geste de zoom	ZoomEvent	Node, Scene
Activation du menu contextuel	ContextMenuEvent	Node, Scene
Texte modifié (durant la saisie)	InputMethodEvent	Node, Scene
Bouton cliqué ComboBox ouverte ou fermée Une des options d'un menu contextuel activée Option de menu activée Pression sur <i>Enter</i> dans un champ texte	ActionEvent	ButtonBase ComboBoxBas ContextMenu MenuItem TextField
Élément (<i>Item</i>) d'une liste, d'une table, d'un arbre a été édité	EditEvent CellEditEvent EditEvent	ListView TableColumn TreeView
Erreur survenue dans le <i>media-player</i>	MediaErrorEvent	MediaView
Menu est affiché (déroulé) ou masqué (enroulé)	Event	Menu
Fenêtre <i>popup</i> masquée	Event	PopupWindow
Onglet sélectionné ou fermé	Event	Tab
Fenêtre affichée, fermée, masquée	WindowEvent	Window

- Chaque objet de type "événement" comprend (au moins) les 3 informations suivantes :
 - Le **type de l'événement** (**EventType**) consultable avec **getEventType()**. Le type permet de classifier les événements à l'intérieur d'une même classe (par exemple, la classe **KeyEvent** englobe **KEY_PRESSED**, **KEY_RELEASED**, **KEY_TYPED**)
 - La **source de l'événement** (**Object** consultable avec **getSource()**) : Objet qui est à l'origine de l'événement selon la position dans la chaîne de traitement des événements (*event dispatch chain*).
 - La **cible de l'événement** (**EventTarget** consultable avec **getTarget()**) : Composant cible de l'événement (indépendamment de la position dans la chaîne de traitement des événements (*event dispatch chain*))
- Chaque type d'événement possède un nom (**getName()**) et un type parent (**getSuperType()**).



GESTION DES ÉVÈNEMENTS

- JavaFx dispose d'une interface fonctionnelle (générique) **EventHandler** `<T extends Event>` qui impose l'unique méthode `handle(T event)` qui se charge de traiter l'événement.

```
button1.setOnAction(new EventHandler<ActionEvent>(){  
    @Override  
    public void handle(ActionEvent event) {  
        txtA.appendText("Test Bouton 1");  
    }  
});
```

```
button1.setOnAction(event-> {txtA.appendText("A");});
```



EXEMPLE CLICK BOUTON : ACTIONEVENT

// code de l'interface sans la gestion des évènements

```
import javafx.application.Application;
import javafx.event.EventHandler;
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.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;

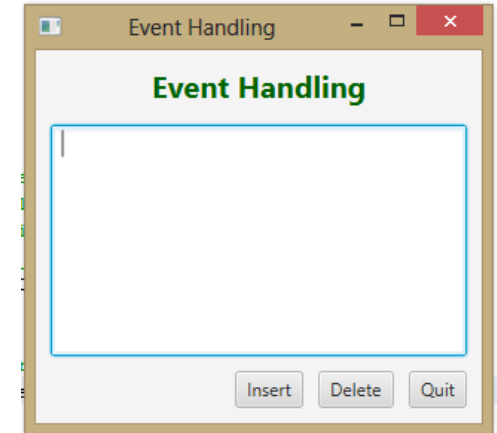
public class EventHandling extends Application {
    private BorderPane root = new BorderPane();
    private HBox btnPanel = new HBox(10);
    private Label lblTitle = new Label("Event Handling");
    private TextArea txtMsg = new TextArea();
    private Button btnInsert = new Button("Insert");
    private Button btnDelete = new Button("Delete");
    private Button btnQuit = new Button("Quit");

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Event Handling");
        root.setPadding(new Insets(10));
        //--- Title
        lblTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
        lblTitle.setTextFill(Color.DARKGREEN);
        BorderPane.setAlignment(lblTitle, Pos.CENTER);
        BorderPane.setMargin(lblTitle, new Insets(0, 0, 10, 0));
        root.setTop(lblTitle);
```

```
        //--- Text-Area
        txtMsg.setWrapText(true);
        txtMsg.setPrefColumnCount(15);
        txtMsg.setPrefRowCount(10);
        root.setCenter(txtMsg);
        //--- Button Panel
        btnPanel.getChildren().add(btnInsert);
        btnPanel.getChildren().add(btnDelete);
        btnPanel.getChildren().add(btnQuit);
        btnPanel.setAlignment(Pos.CENTER_RIGHT);
        btnPanel.setPadding(new Insets(10, 0, 0, 0));
        root.setBottom(btnPanel);
```

```
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

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



- Les évènements :
- ajouter ou supprimer la lettre « A »
 - quitter la plateforme.



EXEMPLE CLICK BOUTON (2)

PARTIE ÉVÉNEMENTIELLE - PREMIER MÉTHODE

// classe anonyme implémentant l'interface EventHandler

```
btnInsert.setAction(new EventHandler<ActionEvent>(){
    @Override
    public void handle(ActionEvent event) {
        txtMsg.appendText("A");
    });

btnDelete.setAction(new EventHandler<ActionEvent>(){
    @Override
    public void handle(ActionEvent event) {
        txtMsg.deletePreviousChar();
    });

btnQuit.setAction(new EventHandler<ActionEvent>(){
    @Override
    public void handle(ActionEvent event) {
        Platform.exit();});
```

// par expression Lambda

```
btnInsert.setAction(event -> ->
    {txtMsg.appendText("A");});

btnDelete.setAction(event -> ->
    {txtMsg.deletePreviousChar();});

btnQuit.setAction(event ->
    {Platform.exit();});
```



DEUXIÈME MÉTHODE :

```
import javafx.event.ActionEvent;    import javafx.event.EventHandler; import javafx.scene.control.TextArea;

public class Controleur implements EventHandler<ActionEvent>{
private TextArea tArea;
//--- Constructeur -----
    public Controleur(TextArea tArea) { this.tArea = tArea;}
//--- Code exécuté lorsque l'événement survient ----
    @Override
    public void handle(ActionEvent event) { tArea.appendText("A");} }

public class Controleur1 implements EventHandler<ActionEvent> {
private TextArea tArea;
//--- Constructeur -----
public Controleur1(TextArea tArea) { this.tArea = tArea;}
//--- Code exécuté lorsque l'événement survient ----
    @Override
    public void handle(ActionEvent event) { tArea.deletePreviousChar(); } }

public class Controleur2 implements EventHandler<ActionEvent>{
//--- Constructeur -----
public Controleur2() {}
//--- Code exécuté lorsque l'événement survient ----
    @Override
    public void handle(ActionEvent event) { Platform.exit();} }

// dans la méthode start()
// partie événementielle
// par controleur
Controleur insertCtrl = new Controleur(txaMsg);
btnInsert.addEventHandler(ActionEvent.ACTION, insertCtrl);
Controleur1 insertCtrl1 = new Controleur1(txaMsg);
btnDelete.addEventHandler(ActionEvent.ACTION, insertCtrl1);
Controleur2 insertCtrl2 = new Controleur2();
btnQuit.addEventHandler(ActionEvent.ACTION, insertCtrl2);
```

Ceci peut se
réduire par un
seul contrôleur



Ceci peut se réduire par un seul contrôleur :

```
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
public class ControleurGlob implements EventHandler<ActionEvent>{
    private TextArea tArea;
    private Button b1,b2,b3;
    //--- Constructeur -----
    public ControleurGlob(TextArea tArea, Button b1, Button b2, Button b3) {
        this.tArea = tArea;
        this.b1=b1;
        this.b2=b2;
        this.b3=b3;
    }
    //--- Code exécuté lorsque l'événement survient ----
    @Override
    public void handle(ActionEvent event) {

        if(event.getSource() instanceof Button){
            Button b = (Button) event.getSource();
            if(b==b1) tArea.appendText("A");
            else if(b==b2) tArea.deletePreviousChar();
            else Platform.exit();
        }
    }

    // dans la méthode start()
    // partie événementielle
    // par un seul controleur
    ControleurGlob insertCtrlGlob = new ControleurGlob(txaMsg,btnInsert,btnDelete,btnQuit);
    btnInsert.addEventHandler(ActionEvent.ACTION, insertCtrlGlob);
    btnDelete.addEventHandler(ActionEvent.ACTION, insertCtrlGlob);
    btnQuit.addEventHandler(ActionEvent.ACTION, insertCtrlGlob);
}
```



Troisième méthode :

Créer le contrôleur sous la forme d'une classe locale anonyme.

```
// partie événementielle
```

```
// contrôleur= classe locale anonyme
```

```
btnInsert.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>() {  
    @Override  
        public void handle(ActionEvent event) {  
            txtA.appendText("A");  
        }  
});
```

```
btnDelete.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>() {  
    @Override  
        public void handle(ActionEvent event) {  
            txtA.deletePreviousChar();  
        }  
});
```

```
btnQuit.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>() {  
    @Override  
        public void handle(ActionEvent event) {  
            Platform.exit();  
        }  
});
```



LES ÉVÉNEMENTS SOURIS : **MOUSEEVENT**

- Il est possible de capturer tous les types d'événement souris existant :
 - Entrée de la souris dans une zone.
 - Sortie de la souris.
 - Clic.
 - Relâchement du clic.
 - Mouvement quelconque de la souris.
 - Mouvement de la roulette de la souris.
 - Glissement de la souris (mouvement pendant un clique).
- Pour détecter chacun de ces événements, la méthode est toujours la même.
 - Chaque objet graphique et chaque groupe d'objets graphiques possède un certain nombre de paramètres auxquels on peut affecter un objet de type *EventHandler* qui exécutera une fonction à chaque fois qu'un certain type d'événement se produit.
 - Par exemple pour déclencher une action à chaque fois qu'on clique sur un objet, il suffit d'initialiser le paramètre `onMouseClicked` avec un `EventHandler`



LES ÉVÉNEMENTS SOURIS

```
import javafx.event.EventHandler;  
import javafx.scene.input.MouseEvent;
```

```
objet.setOnMouseClicked(new EventHandler<MouseEvent>(){  
    public void handle(MouseEvent me){  
        //instructions à exécuter lors de cet événement  
    });
```

// Survol de la souris

```
this.setOnMouseEntered(new EventHandler<MouseEvent>(){  
    public void handle(MouseEvent me){  
        //instructions  
    }    });  
this.setOnMouseExited(new EventHandler<MouseEvent>(){  
    public void handle(MouseEvent me){  
        //instructions  
    }    });
```

- Le paramètre *me* de type `MouseEvent` de la fonction *handle* permet d'avoir toutes sortes d'informations sur l'événement. Cette méthode est très simple et rapide à utiliser
- elle est valable pour tous les nœuds graphiques : les rectangles, les cercles, les textes, les groupes et bien sûr ceux que nous créons

LES ÉVÉNEMENTS CLAVIER : **KEYEVENT**

- Pour permettre à un objet graphique de capturer des événements clavier, la démarche est la même que pour les événements souris.
- Il suffit d'affecter aux paramètres `onKeyPressed`, `onKeyReleased` ou `onKeyTyped`, des objets de type *EventHandler* qui exécuteront une certaine fonction à chaque fois que l'un des événements correspondant sera réalisé :
 - l'utilisateur appuie sur une touche
 - l'utilisateur relâche une touche
- La syntaxe est la même que pour les clics souris



```
import javafx.event.EventHandler;
import javafx.scene.input.KeyEvent;


objet.onKeyPressed (new EventHandler<KeyEvent>(){
    public void handle(KeyEvent ke){
        //instructions à exécuter lors de cet événement
    }
});

objet.onKeyReleased (new EventHandler<KeyEvent>(){
    public void handle(KeyEvent ke){
        //instructions à exécuter lors de cet événement
    }
});
```

- le paramètre *ke* de type `KeyEvent` de la fonction *handle* permet d'avoir des informations sur l'événement, notamment la lettre de la touche grâce à la fonction *ke.getText()*



- Il y a une différence importante entre les **événements souris et les événements clavier**.
 - Tous les éléments graphiques situés dans une fenêtre peuvent être sensibles aux événements souris en même temps.
 - Alors qu'un seul objet graphique à la fois peut être sensible aux événements clavier.

 **On dit que c'est l'objet qui a le focus qui pourra capter les événements clavier.**

- Pour qu'un objet ait le focus, il suffit de le lui attribuer grâce à la fonction :
 - `mon_objet.requestFocus();`
 - On ajout à la fin de la fonction `Start()`



LE CHANGELISTENER

- Un *ChangeListener* est un objet qui peut être ajouté à n'importe quelle propriété de n'importe quel objet. Il permet de détecter un changement de cette propriété et de déclencher une action lors de ce changement
- Une propriété : est un objet qui ne contient qu'une seule variable et des fonctions qui s'appliquent à cette variable.
- Une propriété est une sorte de super-variable
 - Par exemple : **un objet *groupe* de type *ToggleGroup*** contient la propriété *selectedToggle* dont la valeur correspond au **bouton radio sélectionné**.
 - Ajouter un *ChangeListener* à cette propriété
 - la fonction *changed()* s'exécutera à chaque fois que la propriété *selectedToggle* changera de valeur

```
groupe.selectedToggleProperty().addListener(new ChangeListener() {  
    public void changed(ObservableValue observable, Object oldValue, Object newValue) {  
  
        //instructions  
    }  
});
```



LES PROPRIÉTÉS EN JAVAFX

- **une variable** : une case qui permet de stocker une information d'un certain type : int, float, boolean...
 - Pour déclarer une variable : **Type ma_variable;**
 - Pour affecter une valeur : **ma_variable = <valeur>;**
- **une propriété** : C'est un objet qui contient une variable d'un certain type : int, float, boolean...
 - Pour déclarer une propriété de type int on écrit : **IntegerProperty ma_propriete = new IntegerProperty();**
 - Pour Affecter une valeur : **ma_property.set(<valeur>);**
 - Pour lire sa valeur : **ma_property.get();**



À QUOI ÇA SERT D'UTILISER UNE PROPRIÉTÉ PLUTÔT QU'UNE VARIABLE ?

- la seule différence réside dans la façon dont on accède au contenu de la variable, en lecture et en écriture.
- **L'un des principaux intérêts des propriétés est ce qu'on appelle le *data binding*.**



LE DATA BINDING

- En anglais *data* signifie "donnée" et *to bind* signifie "lier".
- Le *data binding* est un mécanisme qui permet de lier plusieurs données entre elles.
- Ce mécanisme ne peut s'appliquer qu'à des propriétés, il est impossible de lier deux variables

"lier" ?



- lie deux propriétés, on désigne toujours une propriété *maitre* et une propriété *esclave*. On dit que la valeur de la propriété *esclave* dépend de celle de la propriété *maitre*.
 - Si par exemple on veut déclarer deux propriétés de type int A et B et si on souhaite lier la propriété B à la propriété A, c'est-à-dire faire de A le maitre et de B l'esclave,
 - Exemple : la valeur de B soit toujours égale à celle de A, s'écrit :

```
IntegerProperty A = new IntegerProperty();  
IntegerProperty B = new IntegerProperty();
```

B.bind(A);//on lie B à A

Dès qu'on modifiera la valeur de A, cela modifiera automatiquement celle de B.

Par contre on ne peut plus modifier directement la valeur de B, pour cela on est obligé de modifier celle de A.

B est donc liée à A, pour le meilleur et pour le pire

- Exemple : Lier B à A de façon à ce que B soit toujours égale au double de A

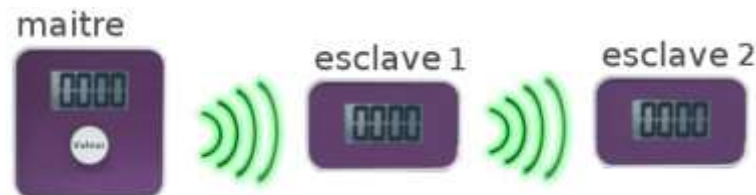
`B.bind(A.multiply(2));`

- Si par exemple on affecte la valeur 2 à A, la valeur de B deviendra $2*2 = 4$.
- La valeur de la propriété *maitre* peut être modifiée et cela induira automatiquement la modification de la valeur de la propriété *esclave*. Par contre la valeur de la propriété *esclave* ne peut plus être modifiée
- On peut ainsi faire toutes les opérations que l'on souhaite sur A

`B.bind(A.multiply(5).add(3.5).divide(2).subtract(0.95));`

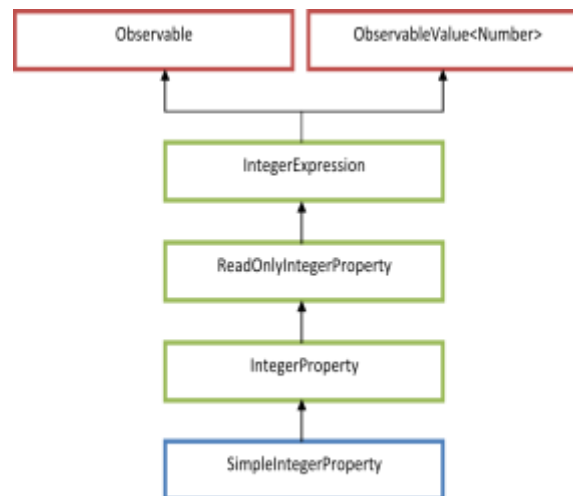
`// B = (A*5 + 3,5) / 2 - 0,95`

lier plusieurs propriétés esclave à une même propriété maitre



LES PROPRIÉTÉS

- La notion de "propriété" (property) est très présente dans JavaFX
 - une propriété est un élément d'une classe que l'on peut manipuler à l'aide de getters (lecture) et de setters (écriture).
- Les propriétés sont généralement représentées par des attributs de la classe mais elles pourraient aussi être stockées dans une base de données ou autre système d'information.
- En plus des méthodes `get...()` et `set...()`, les propriétés JavaFX possèdent une troisième méthode `...Property()` qui retourne un objet qui implémente l'interface `Property` [... : nom de la propriété].
- JavaFX propose une série de classes et d'interfaces dédiées à la définition des propriétés qu'elles soient en lecture seule, en lecture-écriture...
- Toutes les classes de propriétés implémentent l'interface **Observable** et offrent de ce fait, la possibilité d'enregistrer des *observateurs* (**Listener**) qui seront avertis lorsque la valeur de la propriété change.



EXEMPLE

- Une instance de l'interface fonctionnelle `ChangeListener<T>` pourra ainsi être créée pour réagir à un tel changement. La méthode `changed()` sera alors invoquée et recevra en paramètre la valeur observée ainsi que l'ancienne et la nouvelle valeur de la propriété.

- **Exemple : propriété balance dans un compte bancaire**

```
public class BankAccount
{
    private DoubleProperty balance = new SimpleDoubleProperty();
    public final double getBalance() { return balance.get(); }
    public final void setBalance(double amount) { balance.set(amount); }
    public final DoubleProperty balanceProperty() { return balance; }
}

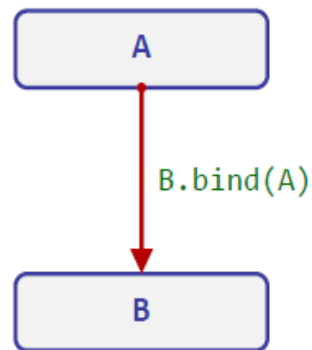
    ○ /* La classe abstraite DoubleProperty permet d'emballer une valeur de type double et d'offrir des
        méthodes pour consulter et modifier la valeur mais également pour "observer" et "lier" les changements*/
    ○ // SimpleDoubleProperty est une classe concrète prédéfinie.
```

- **Continuité de l'exemple**

```
DoubleProperty sum = account.balanceProperty();
/* Dans l'exemple, une expression lambda est utilisée pour implémenter la méthode changed(). On pourrait
également utiliser une classe anonyme ou créer l'instance d'une classe "ordinaire" qui implémente l'interface
ChangeListener<T>*/
.sum.addListener( (ObservableValue <? extends Number> obsVal, Number oldVal, Number newVal) ->
{ //-- ChangeListener code
System.out.println(oldVal+" devient"+ newVal); } );
```

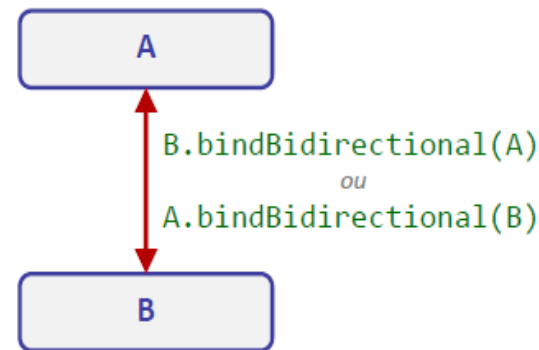


- L'interface fonctionnelle `InvalidationListener<T>` permet également de réagir aux changements des valeurs de propriétés dans les situations où les propriétés sont calculées à partir d'autres et que l'on veut éviter d'effectuer les calculs à chaque changement. Avec cette interface, c'est la méthode `invalidated(Observable o)` qui est invoquée lorsqu'un changement potentiel de la valeur de la propriété est intervenu.
- Un des avantages des propriétés *JavaFX* est la possibilité de pouvoir les lier entre elles. Ce mécanisme, appelé ***binding***, permet de mettre à jour automatiquement une propriété en fonction d'une autre. Dans les interfaces utilisateurs, on a fréquemment ce type de liens. Par exemple, lorsqu'on déplace le curseur d'un *slider*, la valeur d'un champ texte changera (ou la luminosité d'une image, la taille d'un graphique, le niveau sonore...)



Liaison unidirectionnelle

(la valeur de la propriété B dépend de la valeur de la propriété A)



Liaison bidirectionnelle

(la valeur de la propriété B dépend de la valeur de la propriété A et inversement)



BINDING

- Les composants utilisés dans les interfaces graphiques (boutons, champs texte, cases à cocher, *sliders*...) possèdent tous de nombreuses **propriétés** (voir notion de propriété en Java précédemment). Pour lier ces propriétés on utilise la technique de binding.
- Une propriété ne peut être liée (asservie) qu'à une seule autre si le lien est unidirectionnel (`bind()`). Par contre, les liens bidirectionnels (`bindBidirectional()`) peuvent être multiples.
- Parfois, une propriété dépend d'une autre mais avec une relation plus complexe. Il est ainsi possible de créer des ***propriétés calculées***. Deux techniques sont à disposition (elles peuvent être combinées) :
 - utiliser la classe utilitaire **Bindings** qui possède de nombreuses méthodes statiques permettant d'effectuer des opérations.
 - utiliser les méthodes disponibles dans les classes qui représentent les propriétés; ces méthodes peuvent être chaînées (*Fluent API*)
- Des opérations de conversions sont parfois nécessaires si le type des propriétés à lier n'est pas le même. Par exemple pour lier un champ texte (`StringProperty`) à un *slider* dont la valeur est numérique (`DoubleProperty`).



Exemple de Binding : binding une valeur de slider dans un label

```
import javafx.application.Application;      import javafx.stage.Stage;
import javafx.beans.value.ChangeListener;   import javafx.beans.value.ObservableValue;
import javafx.scene.Scene;                  import javafx.scene.control.Label;
import javafx.scene.control.Slider;         import javafx.scene.layout.HBox;
```

```
public class Binding1 extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        final Slider slider = new Slider(0, 100, 50);
        final Label text = new Label();
        text.setText(Math.round(slider.getValue()) + "");
        slider.valueProperty().addListener(new ChangeListener<Number>() {
            @Override
            public void changed(ObservableValue<? extends Number> observableValue, Number oldValue, Number
newValue) {
                if (newValue == null) {
                    text.setText("");
                    return;
                }
                text.setText(Math.round(newValue.intValue()) + "");
            }
        });
        text.setPrefWidth(50);
        HBox root = new HBox(2);
        root.getChildren().addAll(slider, text);
        Scene scene = new Scene(root, 200, 50);
        primaryStage.setTitle("Binding");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



A series of five orange circles of varying sizes are arranged in a vertical line on the left side of the slide. The largest circle is at the top, followed by a medium-sized circle, then a small circle, then another medium-sized circle, and finally a small circle at the bottom.

COURS JAVA AVANCÉ

IMPLÉMENTATION DU MVC EN JAVA

Mme Nouria Sana

ARCHITECTURE MVC

- L'intégration des contrôleurs dans les exemples précédents nous fait joindre à une architecture de structure des applications interactives qui entre dans la catégorie des design-pattern :
 - l'architecture MVC (Model-View-Controller)
- Dans cette architecture on divise le code des applications en entités distinctes (***modèles, vues et contrôleurs***) qui communiquent entre elles au moyen de divers mécanismes (invocation de méthodes, génération et réception d'événements, etc.).
 - Le **modèle** (***Model***) est responsable de la gestion des données qui caractérisent l'état du système et son évolution. Il est souvent défini par une ou plusieurs interfaces *Java* qui permettent de s'abstraire de la façon dont les données (les objets *métier*) sont réellement stockées. Il offre également les méthodes et fonctions permettant de gérer, transformer et manipuler ces données.
 - La **vue** (***View***) est chargée de la représentation visuelle des informations en faisant appel à des écrans, des fenêtres, des composants, des conteneurs (*layout*), des boîtes de dialogue, etc. Plusieurs vues différentes peuvent être basées sur le même modèle.
 - Le **contrôleur** (***Controller***) est chargé de réagir aux différentes actions de l'utilisateur ou à d'autres événements qui peuvent survenir. Dans les applications simples, il gère la synchronisation entre la vue et le modèle (rôle de chef d'orchestre).



- Une application *JavaFX* qui respecte l'architecture MVC comprendra généralement différentes classes et ressources :
 - Le **modèle** sera fréquemment représenté par une ou plusieurs classes qui implémentent généralement une interface permettant de s'abstraire des techniques de stockage des données.
 - Les **vues** seront soit codées en *Java* ou déclarées en FXML. Des feuilles de styles CSS pourront également être définies pour décrire le rendu.
 - Les **contrôleurs** pourront prendre différentes formes : Ils peuvent être représentés par des classes qui traitent chacune un événement particulier ou qui traitent plusieurs événements en relation (menu ou groupe de boutons par exemple). Si le code est très court, ils peuvent parfois être inclus dans les vues, sous forme de classes locales anonymes ou d'expressions lambda.
- La **classe principale** (celle qui comprend la méthode `main()`) peut faire l'objet d'une classe séparée ou être intégrée à la classe de la fenêtre principale (vue principale).
- D'autres **classes utilitaires** peuvent venir compléter l'application.



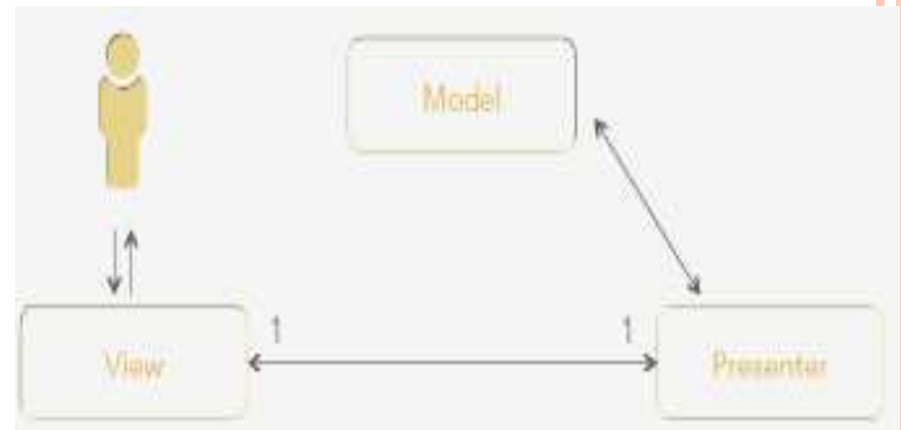
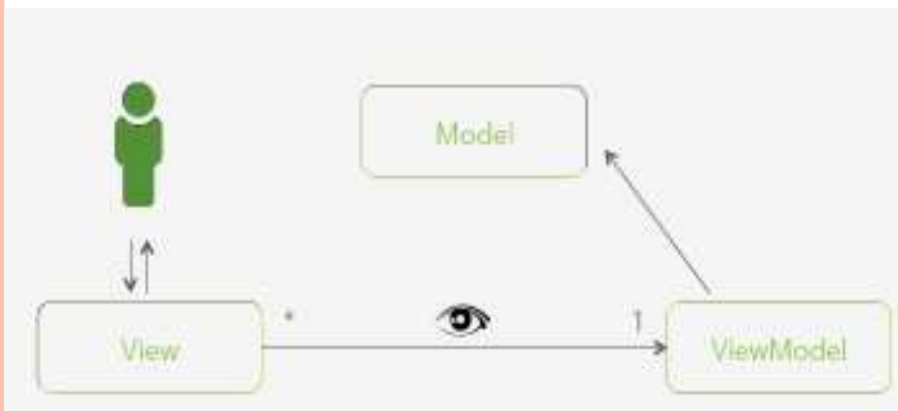
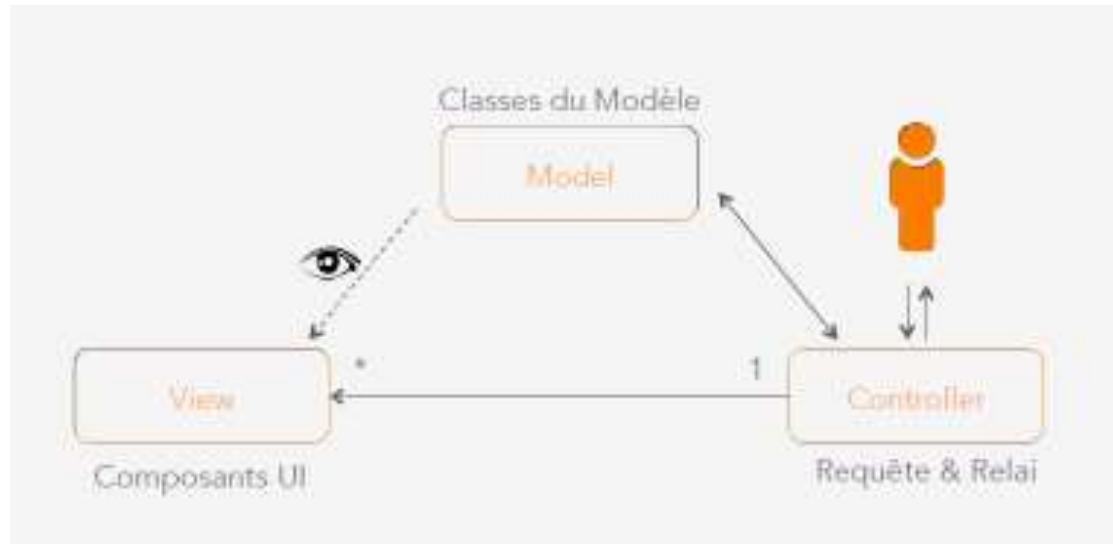
LES PATRONS DE CONCEPTION MV*

○ Avantage

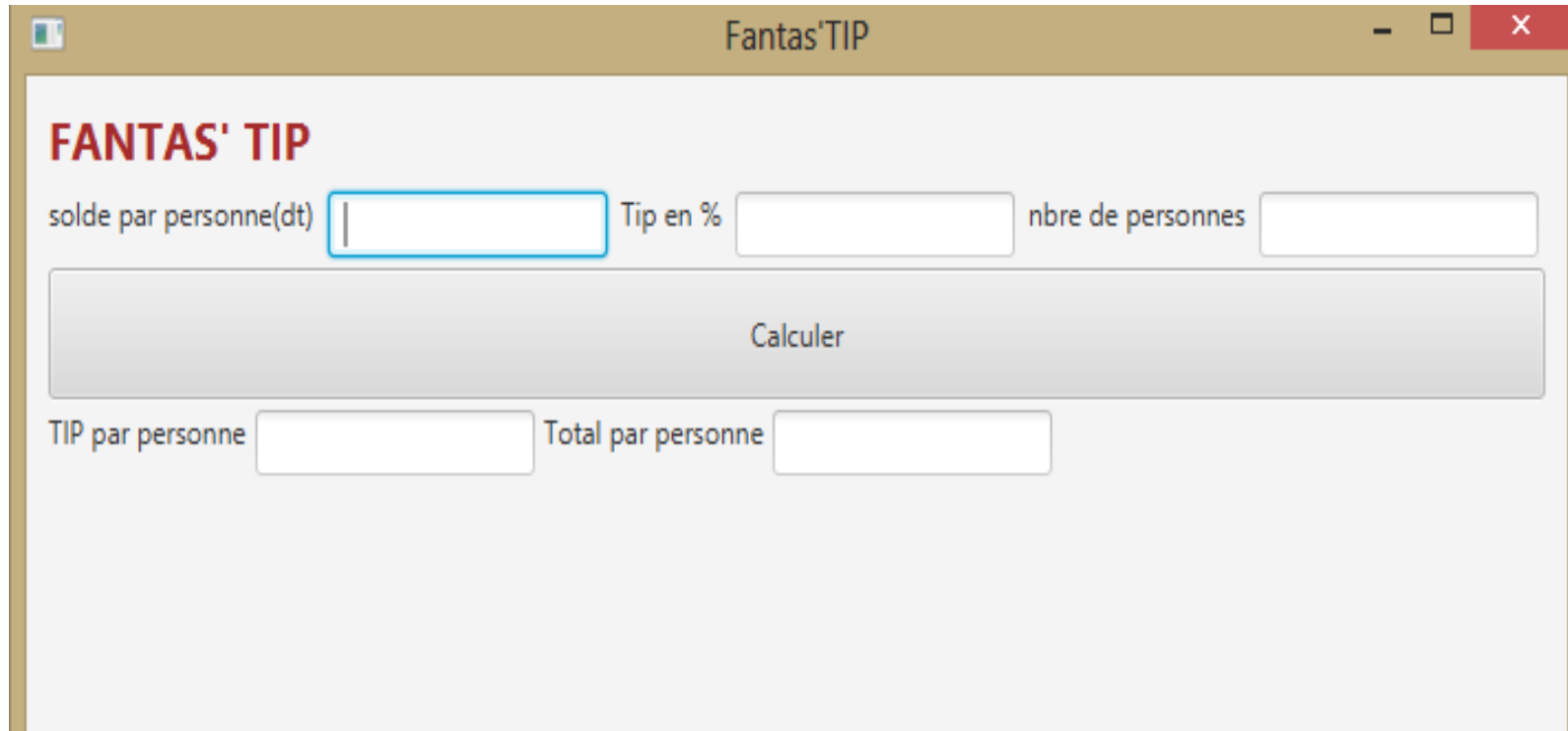
« Les patrons de conception sont des solutions réutilisables pour des problème récurrents »

- Réutilisabilité
 - Le modèle n'est pas couplé avec sa représentation, et peut donc être facilement réutilisé dans d'autres projets
- Testabilité
 - Tester les couche indépendamment les unes des autres les rend plus faciles à gérer et à corriger
- Maintenabilité
 - Il est plus facile de modifier une partie de l'application sans impacter les autres modules
- Compréhensibilité
 - Le code est plus compréhensible et plus lisible

- MVC : Modele Viem Control
- MVVM : Model- View ViewModel
- MVP : Model –View-Present



Exemple : L'application *Fantas'TIP* est un utilitaire qui calcule le pourboire à prévoir et le montant par personne, en fonction du montant de la note, du pourcentage octroyé et du nombre de convives.



The image shows a screenshot of a Windows application window titled "Fantas'TIP". The window has a gold-colored title bar with standard minimize, maximize, and close buttons. The main content area has a light gray background. At the top left, the text "FANTAS' TIP" is displayed in a bold, dark red font. Below this, there are three input fields: "solde par personne(dt)" followed by a text box with a blue border, "Tip en %" followed by a text box, and "nbre de personnes" followed by a text box. A large, light gray button labeled "Calculer" is centered below these fields. At the bottom, there are two more text boxes: "TIP par personne" and "Total par personne".

// Le modèle

```
public interface InterfaceModèle {  
    void setSolde(double solde);  
    void setTipPercent(int percent);  
    void setNbPersonne(int nbPersonne);  
    double getTipParPersonne();  
    double getTotalParPersonne(); }  

```

```
public class Modèle implements InterfaceModèle {  
    private double solde = 0;    private int percent = 0;    private int nbPersonne = 1;  
    public Modèle() { }  
    @Override  
    public void setSolde(double solde) {  
        if (solde < 0) throw new IllegalArgumentException("solde < 0");  
        this.solde = solde; }  
    @Override  
    public void setTipPercent(int percent) {  
        if (percent < 0) throw new IllegalArgumentException("Percent < 0");  
        this.percent = percent; }  
    @Override  
    public void setNbPersonne(int nbPersonne) {  
        if (nbPersonne <= 0) throw new IllegalArgumentException("Nb personnes <= 0");  
        this.nbPersonne = nbPersonne; }  
    @Override  
    public double getTipParPersonne() {  
        return solde * percent / 100.0 / nbPersonne; }  
    @Override  
    public double getTotalParPersonne() {  
        return solde/nbPersonne + getTipParPersonne(); }  
}
```



// La vue

```
import javafx.application.Application; import javafx.geometry.Insets; import javafx.scene.text.FontWeight;
import javafx.geometry.Pos;          import javafx.scene.Scene; import javafx.stage.Stage; import javafx.scene.layout.VBox;
import javafx.scene.control.Button;   import javafx.scene.control.Label; import javafx.scene.text.Font;
import javafx.scene.control.TextField; import javafx.scene.layout.HBox; import javafx.scene.paint.Color;
```

public class **FantasTip1** extends Application {

```
VBox root; HBox hb1, hb2, hb3, hb4;
Label lb1, lb2, lb3, lb4, lb5, lb6;
TextField t1, t2, t3, t4, t5; Button b;
```

Modèle md = new Modèle();

@Override

```
public void start(Stage primaryStage) {
    root = new VBox(4);
    root.setPadding(new Insets(10));
    hb1 = new HBox(1);
    lb1 = new Label ("FANTAS' TIP");
    lb1.setFont(Font.font("System", FontWeight.BOLD, 20));
    lb1.setTextFill(Color.BROWN);    lb1.setAlignment(Pos.CENTER);
    hb1.getChildren().add(lb1);
    lb2 = new Label ("solde par personne(dt)");
    t1 = new TextField();    t1.setPrefColumnCount(10);
    lb3 = new Label ("Tip en %");
    t2 = new TextField();    t2.setPrefColumnCount(10);
    lb4 = new Label ("nbre de personnes");
    t3 = new TextField();    t3.setPrefColumnCount(10);
    hb2 = new HBox(6);
    hb2.getChildren().addAll(lb2, t1, lb3, t2, lb4, t3);
    b = new Button("Calculer");
    b.setPrefSize(700, 50);
    b.setAlignment(Pos.CENTER);
    hb3 = new HBox(1);    hb3.getChildren().add(b);
    lb5 = new Label ("TIP par personne");
    t4 = new TextField();    t4.setPrefColumnCount(10);
    lb6 = new Label ("Total par personne");
    t5 = new TextField();    t5.setPrefColumnCount(10);
    hb4 = new HBox(4);
    hb4.getChildren().addAll(lb5, t4, lb6, t5);
    root.getChildren().addAll(hb1, hb2, hb3, hb4);
}
```

// partie événementielle

Controleur cl = new Controleur(this, md);

b.setOnAction(cl);

Scene scene = new Scene(root, 700, 250);

primaryStage.setTitle("Fantas'TIP");

primaryStage.setScene(scene);

primaryStage.show(); }

double getSolde(){

return Double.valueOf(t1.getText()); }

int getTipPercent(){

return Integer.valueOf(t2.getText()); }

int getNbPersonnes(){

return Integer.valueOf(t3.getText()); }

void setTipParPersonne(double tpp){

t4.setText(Double.toString(tpp)); }

void setTotalParPersonne(double tpp){

t5.setText(Double.toString(tpp)); }

public static void main(String[] args) {

launch(args); }



// Le controleur

```
import javafx.event.ActionEvent;
```

```
import javafx.event.EventHandler;
```

```
public class Controleur implements EventHandler <ActionEvent>{
```

```
    FantasTip1 ft; // vue
```

```
    Modèle md; // modèle
```

```
    Controleur(FantasTip1 ft, Modèle md){
```

```
        this.ft = ft;    this.md = md;}
```

```
    @Override
```

```
    public void handle(ActionEvent event){
```

```
        try {
```

```
            double solde = ft.getSolde();
```

```
            int tipPercent = ft.getTipPercent();
```

```
            int nbPersonne = ft.getNbPersonnes();
```

```
            md.setSolde(solde);
```

```
            md.setTipPercent(tipPercent);
```

```
            md.setNbPersonne(nbPersonne);}
```

```
        catch (IllegalStateException e) { // Erreurs dans certaines valeurs d'entrée
```

```
            return;}
```

```
            ft.setTipParPersonne(md.getTipParPersonne());
```

```
            ft.setTotalParPersonne(md.getTotalParPersonne()); }
```

```
    }
```



EXEMPLE 2 : MVCRESTAURANT AVEC ENREGISTREMENT SUR FICHER

FantasTip

Données d'un groupe

Numéro du groupe

Nombre de personnes

Solde Tip pourcentage

Calcul pour ce groupe:
Tip par personne= 1.0
Total par personne= 11.0

Contenu du fichier

groupe:4.0 Nbr personnes = 5.0 Solde = 50.0 Tip percent = 10.0



PROGRAMME PRINCIPAL

```
import java.io.File;
import java.io.IOException;
import javafx.application.Application;

public class MVCRestaurant {

    public static void main(String[] args) throws IOException {
        File fichier = new File ("restaurant.txt");
        if (!fichier.exists()) fichier.createNewFile();
        Application.launch(Vue.class, args);
    }

}
```



MODELE

```
import java.io.Serializable;
```

```
public class Modele implements Serializable {  
    private double cle = 1;  
    private double solde = 0;  
    private double percent = 0;  
    private double nbPersonne = 1;  
    // constructeur  
    public Modele() {}  
  
    // getters-setters des attributs de la classe  
    public double getcle()  
    {return cle;}  
  
    public void setcle(double cle) {  
        this.cle = cle;}  
  
    public double getSolde()  
    {return solde;}  
  
    public void setSolde(double solde) {  
        if (solde < 0)  
            throw new IllegalArgumentException("solde < 0");  
        this.solde = solde;}  
}
```

```
    public double getTipPercent()  
    {return percent;}  
  
    public void setTipPercent(double percent) {  
        if (percent < 0)  
            throw new IllegalArgumentException("Percent < 0");  
        this.percent = percent;}  
  
    public double getNbPersonne()  
    {return nbPersonne;}  
  
    public void setNbPersonne(double nbPersonne) {  
        if (nbPersonne <= 0)  
            throw new IllegalArgumentException("Nb personnes <=  
0");  
        this.nbPersonne = nbPersonne;}  
  
    // méthodes de calcul  
  
    public double TipParPersonne() {  
        return solde * percent / 100.0 / nbPersonne;}  
  
    public double TotalParPersonne() {  
        return solde/nbPersonne + TipParPersonne();}  
}
```



VUE

```
public class Vue extends Application{
    TextField t1,t2,t3,t4;
    Label tx;
    @Override
    public void start(Stage primaryStage) throws Exception {
        GridPane root = new GridPane();
        root.setAlignment(Pos.CENTER);
        root.setVgap(20);
        root.setHgap(20);
```

```
// vue de l'écriture dans un fichier
    Label lb1 = new Label("Données d'un groupe");
    Label lb2 = new Label("Numéro du groupe");
    t1 = new TextField();
    t1.setPrefColumnCount(2);
    Label lb3 = new Label("Nombre de personnes");
    t2 = new TextField();
    t2.setPrefColumnCount(2);
    HBox hb1 = new HBox();
    hb1.getChildren().addAll(lb2,t1);
    HBox hb2 = new HBox();
    hb2.getChildren().addAll(lb3,t2);
    Label lb4 = new Label("Solde");
    t3 = new TextField();
    t3.setPrefColumnCount(2);
    Label lb5 = new Label("Tip pourcentage");
    t4 = new TextField();
    t4.setPrefColumnCount(2);
    HBox hb3 = new HBox();
    hb3.getChildren().addAll(lb4,t3,lb5,t4);
    Button bt = new Button("Afficher et enregistrer");
    tx = new Label();
    tx.setMinSize(200,200);
    Border border = new Border(new BorderStroke(Color.BLACK,
    BorderStrokeStyle.SOLID, CornerRadii.EMPTY, new BorderWidths(2), new
    Insets(0) ));
```

```
    tx.setBorder(border);
```

```
    Button bk = new Button("contenu du fichier");
```

```
    root.add(lb1,0,0);
        root.add(hb1,0,1);
        root.add(hb2,0,2);
        root.add(hb3,0,3);
        root.add(bt,0,4);
        root.add(tx, 0, 5);
        root.add(bk, 1, 2);
        Modele md = new Modele();
        Controleur ct = new Controleur(this,md);
        bt.setOnAction(ct);
        ControleurLecture cl = new ControleurLecture(this,md);
        bk.setOnAction(cl);
```

```
        Scene sn = new Scene(root,500,500);
        primaryStage.setTitle("FantasTip");
        primaryStage.setScene(sn);
        primaryStage.show();
    }
```

```
    double getNumeroGroupe(){
        return Double.valueOf(t1.getText());
    }
```

```
    double getSolde(){
        return Double.valueOf(t3.getText()); }
    double getTipPercent(){
```

```
        return Double.valueOf(t4.getText()); }
    double getNbPersonnes(){
```

```
        return Double.valueOf(t2.getText()); }
    void setInfor(String str1,String str2){
```

```
        tx.setText("Calcul pour ce groupe:\n Tip par personne=
"+str1+"\nTotal par personne= "+str2);
    }
```



CONTROLEUR

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class Controleur implements EventHandler <ActionEvent>
{
    Vue vue; // vue
    Modele md; // modèle
    Controleur(Vue vue, Modele md){
        this.vue = vue;
        this.md = md;}

    @Override
    public void handle(ActionEvent event) {
        md.setcle(vue.getNumeroGroupe());
        md.setNbPersonne(vue.getNbPersonnes());
        md.setTipPercent(vue.getTipPercent());
        md.setSolde(vue.getSolde());
        String str1= Double.toString(md.TipParPersonne());
        String str2= Double.toString(md.TotalParPersonne());
        vue.setInfor(str1, str2);
        // ecrire dans le fichier
        File fichier1 = new File ("retaurant.txt");
```

```
//Ecrire dans le fichier
        try{
            FileOutputStream fichierOut = new
            FileOutputStream(fichier1);
            ObjectOutputStream oos = new
            ObjectOutputStream(fichierOut);
            oos.writeObject(md);
            oos.flush();

            if (oos != null) {
                oos.flush();
                oos.close();} }
        catch(IOException ev){
            ev.printStackTrace();
        }
    }
}
```



CONTROLEUR

```
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;
import java.util.Collection;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ControleurLecture implements EventHandler
<ActionEvent>{
    Vue vue; // vue
    Modele md; // modèle
    Collection<Label> l;
    ControleurLecture( Vue vue, Modele md){
        this.vue = vue;
        this.md = md;
        l = new ArrayList();
    }
    @Override
    public void handle(ActionEvent event) {
        Stage secondaryStage = new Stage();
        VBox vb = new VBox();
        // ecrire dans le fichier
        File fichier2 = new File ("restaurant.txt");
        // lecture du fichier
```

```
try{
    ObjectInputStream entree = new ObjectInputStream(new
    FileInputStream(fichier2)) ;
    boolean eof=false;
    String str;
    Modele m;
    while (!eof){
        try{
            try{
                m = (Modele) entree.readObject();
                str = "groupe:"+ Double.toString(m.getcle())+" Nbr
                personnes = "+ Double.toString(m.getNbPersonne())
                +" Solde = "+ Double.toString(m.getSolde())+" Tip
                percent = "+ Double.toString(m.getTipPercent());
                l.add(new Label(str));
            }
            catch(ClassNotFoundException ev) {}
        }
        catch(EOFException e){eof=true;}
    }
    catch(IOException ioe){
        ioe.printStackTrace();
    }

    vb.getChildren().addAll(l);

    Scene s2 = new Scene(vb,500,500);
    secondaryStage.setScene(s2);
    secondaryStage.setTitle("Contenu du fichier");
    secondaryStage.show();
}
}
```

