Interface Hommes/Systèmes

I - JavaFx

- - - - - - - - - -

- - - - - - - - - -

ENSMA A3-S5 - période A

2023-2024

M. Richard
richardm@ensma.fr

Sommaire

I - Problématique

1 – Généralités
2 – Le problème
3 – Quelles solutions ?
4 – Quels modèles ?

↪ IHM : 2 définitions

IHM : Interaction Homme Machine

- ≪ *désigne l'ensemble des phénomènes physiques et cognitifs qui interviennent dans la réalisation de tâches avec le concours de l'ordinateur* ≫
- Techniques d'interaction
  - Click, Double Click
  - Drag & Drop
  - Pick & Drop
  - Geste
  - Multimodalité
    - association de différentes techniques d'interaction

Ou ... IHM : Interface Homme Machine

- ≪ *désigne un assemblage de composants logiciels et matériels qui permet l'accomplissement de tâches avec le concours de l'ordinateur* ≫
- Techniques de programmation
  - Fenêtres
    - graphique
  - Événements
  - Modèle d'architecture
  - Boite à outils
    - toolkit

↪ Le problème : Informatique classique VS IHM

Différence fondamentale entre l'informatique classique et l'interaction homme-machine :

- Informatique classique :
  - L'homme s'adapte aux capacités de l'ordinateur
  - L'ordinateur est :
    - déterministe
    - de faible complexité

- Interaction homme-machine :
  - La machine doit s'adapter au comportement de l'homme
  - L'homme est :
    - non déterministe,
    - complexe,
    - variable (individus, temps)

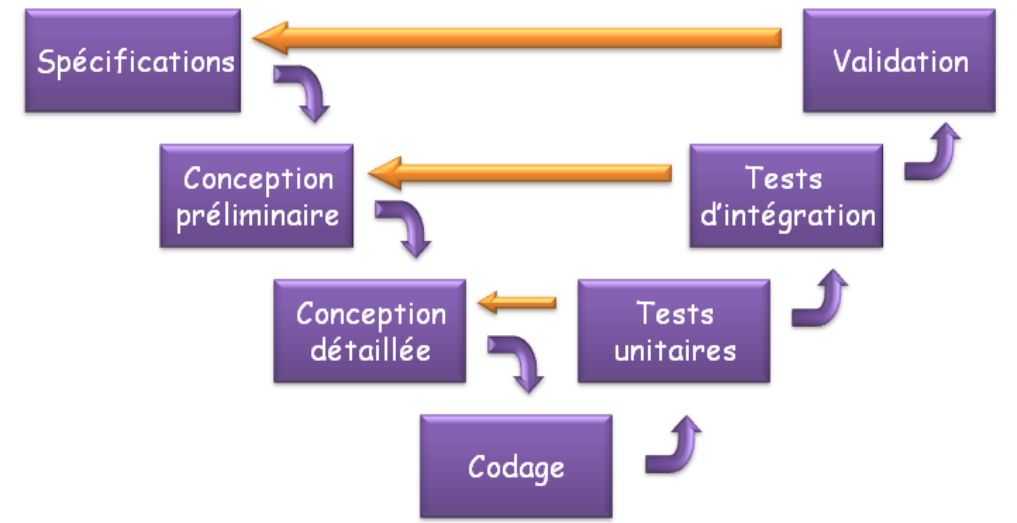
↪ Le problème : Les risques . . .

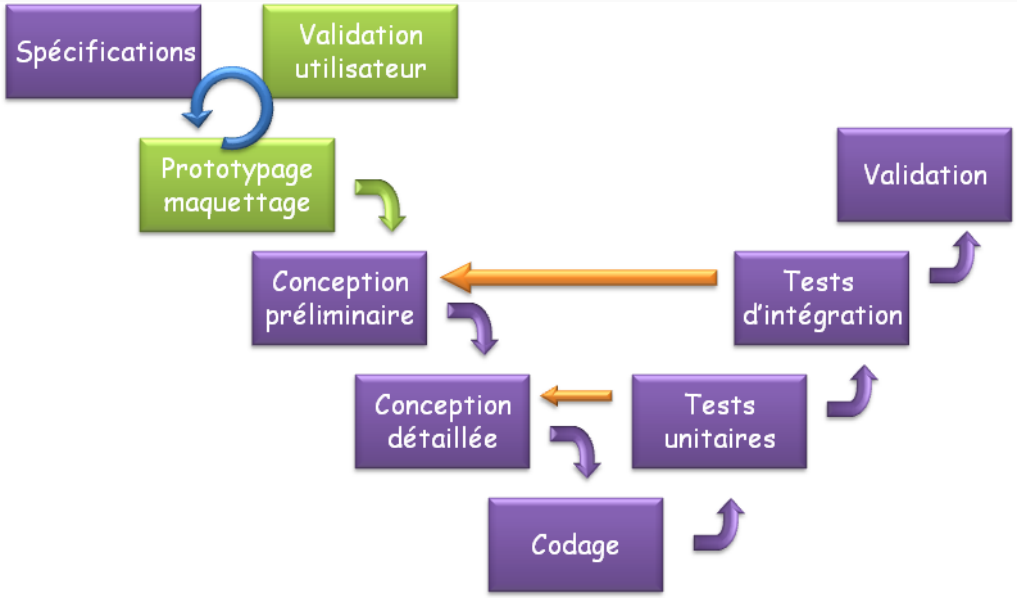
- *Risques associés à une mauvaise IHM :*
  - Application rejetée par les utilisateurs
  - Frustration des utilisateurs,
  - Coût d'apprentissage
    - argent,
    - temps
  - Perte de productivité,
  - Utilisation à 50% (parfois moins) des capacités de l'application
  - Danger physique
    - application critique
    - pilotage avion, train, . . .
    - contrôle aérien,
    - centrale nucléaire,
    - ...

↪ Le problème : en résumé . . .

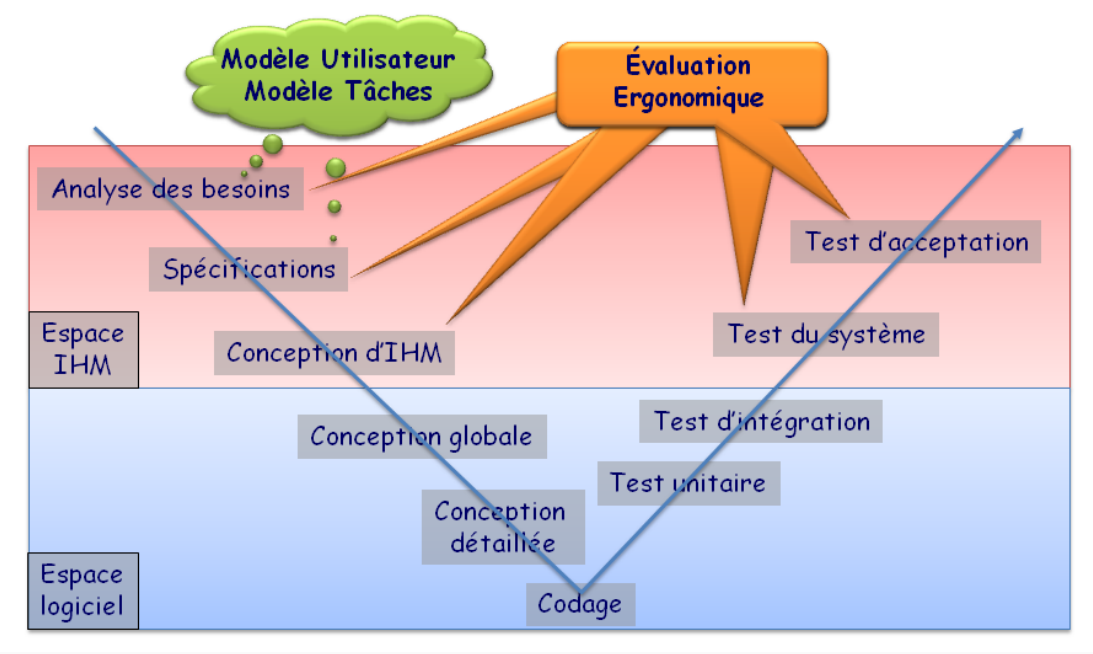
- *En général, dans le domaine du génie logiciel classique, les modèles du cycle de vie s'intéressent à :*
  - La faisabilité :
    - difficulté et travail nécessaire à la réalisation
  - La qualité du logiciel
    - évolutivité,
    - correction,
    - sûreté,
    - maintenance,
    - . . .
- *Dans le domaine des Systèmes Interactifs :*
  - Les points précédents
  - + Utilisabilité
    - facilité d'utilisation par un utilisateur
    - on parle aussi d'acceptabilité

↪ Modification du cycle en V 1/2





↪ Modification du cycle en V 2/2



- Étapes de conception :
  - Analyse des besoins,
  - Analyse des tâches utilisateurs
  - Spécifications du système interactif,
  - Spécifications logicielles
  - Évaluation
- Ces cinq étapes pourront également être incluses dans un cycle itératif.

↪ De nouveaux modèles . . . oui mais . . .

Le domaine de l'Interaction Home-Machine a engendré un grand nombre de formalismes, modèles ou notations permettant de :

- Spécifier tout ou partie de l'interface
- Générer tout ou partie de l'application correspondante

Pourquoi tant de nouveaux formalismes, modèles ou notations ?

- Domaine récent et en plein développement
- C'est un domaine pluridisciplinaire :
  - facteurs humains
    - psychologie, sociologie, ergonomie, . . .
  - aspects informatiques
    - génie logiciel, langages, systèmes, . . .
  - aspect graphiques
    - ergonomie, design, ...
- Ces différents domaines sont (très) difficiles à concilier

Pour toutes ces raisons, les méthodes du génie logiciel classique se prêtent mal au domaine de l'Interaction Homme-Machine.

↪ Modèles, Formalismes & Notations

- *Les modèles, formalismes et notations utilisés peuvent être répertoriés en trois catégories, en fonction des théories desquelles ils sont issus :*

  - Des sciences cognitives,
    - Spécification utilisateur
  - De la théorie des graphes,
    - Spécification utilisateur et dialogue
  - Des approches algébriques/logiques
    - spécification dialogue

II - Bases de JavaFx

1 – Historique des BAO Java
2 – Tour d'horizon . . .
3 – Construction graphique
4 – Composants & Événements

↪ Définition

Utilisation d'une boite à outils (BAO) graphique :

- Une boite à outils est une bibliothèque de composants logiciels permettant de réaliser une présentation graphique des informations et de prendre en compte les actions de l'utilisateur.

Application d'un modèle d'architecture logiciel :

- un modèle d'architecture permet d'organiser les compétences respectives des différents modules logiciels du système interactif
- l'utilisation des composants de la boite à outils doit être limitée à certains modules de l'architecture logicielle

Les choix lors de la mise en œuvre de BAO influent fortement sur les distances articulatoires (Modèle de Norman) :

- difficulté à utiliser les commandes
  - sens de déplacement de l'ascenseur d'une fenêtre
- difficulté de lecture d'un affichage
  - taille ou couleur des caractères inappropriées

↪ Fonctionnement

Affichage des informations vers l'utilisateur :

- usage de composants de présentation (Widgets) :
  - fenêtres,
  - zones graphiques,
  - zone de texte.
- autres modalités :
  - son,
  - voix,
  - retours d'effort,
- ce sont les informations en *sortie*

Prise en compte des actions de l'utilisateur :

- traitement des évènements générés par le système interactif à chaque action de l'utilisateur :
  - déplacement de la souris,
  - clics de la souris,
  - enfoncement d'une touche
- autre modalité :
  - geste,
  - parole
- ce sont les informations en *entrée*

↪ État des lieux

Les BAO actuelles permettent de réaliser des IHM de type WIMP (Windows, Icons, Menus, Pointing devices)
Proposées dans la majorité des cas avec un constructeur d'interface graphique (GUI Builder)

- dispose lui-même d'une interface graphique

Elles sont assez démunies face aux nouvelles modalités d'interaction

- nouveaux widgets : palette transparente, menus circulaires
- nouveaux dispositifs de sortie : retour d'effort, afficheur braille
- reconnaissance de médias continus : parole, écriture manuscrite

Exemples de BAO :

- Swing avec Java (jusqu'à la version 1.7)
- SWT/Java (Eclipse)
- JavaFx avec Java (depuis la version 1.8)
- Android SDK ...

↪ Principaux composants

Les Windows :

- fenêtres de base et boîtes de dialogue

Les Containers :

- composants particuliers permettant de recevoir d'autres composants
- parfois invisibles

Les Components :

- principaux composants d'interaction
- boutons, menus, ...
  - certains utilisent l'architecture MVC (Cf. architecture IHM)

Les LayoutManagers :

- agents de placements des Components dans les Containers
- fixent les règles permettant l'agencement des composants les uns par rapport aux autres à l'intérieur d'un Container

↪ JFC - AWT/Swing/JavaFx 1/2

Les Java Foundation Classes :

- bibliothèques de classes Java
- regroupe un ensemble d'objets permettant d'interagir avec l'utilisateur
  - Composant AWT
    - objets graphiques basiques hérités des anciennes version de Java (Fenêtres, conteneurs, ...)
  - Composants Swing
    - objets graphiques (widgets) plus élaborés que ceux de l'AWT

AWT (Abstract Window Toolkit) :

- BAO historique de Java depuis le JDK 1.0 mais toujours utilisable
- composants implantés à partir des composants natifs des systèmes hôtes (Heavyweight Components)
- *Affichage et comportement* de l'IHM sont *fortement liés au système hôte*

SWING :

- BAO considérée comme standard en Java2 Platform (depuis le JDK 1.2 et jusqu'à la version 1.7)
- pas d'appel aux composants natifs du système hôte
- affichage et comportement indépendants du système hôte
  - ne dépendent que du programmeur ou de l'utilisateur (Pluggable Look and Feel)

↪ JFC - AWT/Swing/JavaFx 2/2

JavaFx :

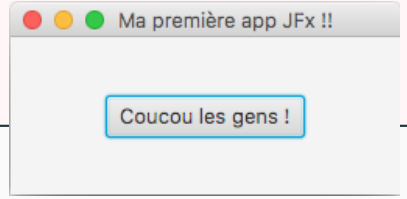
- 2008 (Java 1.6) - JavaFX 1 : tentative (ratée) de remplacement de la BAO Swing
  - basé sur un langage de script (JavaFX Script)
- 2015 (Java 1.8) - JavaFX 8 : une nouvelle version complètement re-développée :
  - Intégrer dans la distribution standard Java (JDK, JRE)
  - Outil d'édition graphique de GUI basé sur XML (FXML) (Scene Builder 2)
  - Réel cohabitation possible avec les composants Swing
  - Nouveaux éléments graphiques :
    - nouvelle API de gestion d'impression
    - nouveaux composants riches (DatePicker, TreeTableView, ...)
    - gestion des écrans tactiles (TouchEvent, GestureEvent, ...)
    - grande amélioration des librairies graphiques 2D et 3D
    - ...
- 2018 (Java 11 (LTS)) - JavaFX 11 : version LTS apportant certaines améliorations
- 2021 (Java 17 (LTS)) - JavaFX 17

↪ Principes

- *Découplage fort entre la partie graphique (FXML + GUI)*
  - conception graphique assurée par un spécialiste (UI designer)
    - pas nécessaire de maîtriser le langage Java et la conception de logiciel
- *Grandes possibilités et souplesse de la partie graphique*
  - utilisation possible de feuilles de style CSS
  - composants complexes dans l'API pour la création d'applications riches :
    - Effets visuels (ombrages, transitions, animations, ...)
    - Graphiques 2D (charts)
    - Navigateur web (WebKit)
    - Images, audio, vidéo (media player)
- *Deux méthodes de créations de la partie graphique d'une interface :*
  - méthode déclarative :
    - déscription de l'interface dans un fichier FXML (syntaxe XML)
    - utilisation de l'utilitaire graphique Scene Builder
  - méthode procédurale :
    - classique, basée sur l'API pour construire l'interface avec du code Java
    - permet par exemple la création de composants étendus
  - Remarque : les deux méthodes peuvent être utilisées conjointement
- *Plateforme d'exécution & portabilité :*
  - application autonome (standalone/desktop application)
  - sur un serveur et intégrée dans une page web (Java Web Start)
  - *en développement :* projet en cours pour le déploiement sur terminaux mobiles (smartphones et tablettes Android, iOS, ...) et sur des systèmes embarqués (Raspberry Pi, ...)

↪ HelloWorld

```
1   package application;
2
3   import javafx.application.Application;
4   import javafx.stage.Stage;
5   import javafx.scene.Scene;
6   import javafx.scene.control.Button;
7   import javafx.scene.layout.BorderPane;
8
9   public class HelloWorld extends Application {
10
11    private Scene scene;
12    private BorderPane root;
13    private Button btnCoucou;
14
15    @Override
16    public void start(Stage primaryStage) {
17      primaryStage.setTitle("Ma première app JFx !!");
18      root = new BorderPane();
19      btnCoucou = new Button("Coucou les gens !");
20      root.setCenter(btnCoucou);
21      scene = new Scene(root, 250, 100);
22      primaryStage.setScene(scene);
23      primaryStage.show();
24    }
25
26    public static void main(String[] args) {
27      launch(args);
28    }
29  }
```



↪ HelloWorld (lancement)

Déroulement de l'application :

- utilisation classique de la méthode `main ...` contenant la méthode statique `Application.launch()`
- méthode `launch()` : le runtime JavaFX effectue les opérations suivantes :
  1. instanciation de la classe héritant de Application
  2. appel de la méthode `init()`
  3. appel de la méthode `start()`
     - en lui passant une instance de `Stage` (correspond à la fenêtre principale)
  4. attente de la fin de l'application
     - soit implicitement(fermeture de la dernière fenêtre)
       `Platform.isImplicitExit()` retourne `true`, soit un appel explicite à `Platform.exit()`
  5. appel de la méthode `stop()`

↪ HelloWorld (lancement (suite))

Redéfinition des méthodes :

- `start()` est abstraite et doit toujours être redéfinie
- `init()` et `stop()` : pas obligatoire (ne font rien par défaut)

Thread d'exécution :

- `start()` et `stop` s'exécute dans le JavaFX Application Thread.
  - dans ce thread que doit être construite l'interface
  - les opérations sur des composants doivent être réalisée dans ce Thread
- `init()` et l'instanciation de la classe qui hérite de `Application` se font dans le thread JavaFX Launcher
  - instanciation de composants graphiques possible, mais pas le placement des composants dans la scene.

↪ Architecture graphique : graphe de scène

Définition du graphe de scène :

- représente la structure hiérarchique de l'interface graphique
- composé de :
  - une racine
  - des nœuds
  - des arcs modélisant les relations parent-enfant
- trois types de nœuds :
  - Racine (unique)
  - Nœud intermédiaire
  - Feuille

Relation Nœuds/Composants graphiques :

- les feuilles sont des composants visibles (boutons, champs texte, ...)
- les nœud intermédiaire (y compris la racine) sont des composants de structuration
  - conteneurs ou panneaux de différents types (`HBox`, `VBox`, `BorderPane`, ...)

↪ Architecture graphique : graphe de scène (suite)

Éléments de l'API

- Tous les éléments d'un graphe de scène sont des instances héritant de la classe `Node`
- De nombreuses classes héritent de `Node`. On distingue 4 grandes familles :
  - les formes primitives de dessin (Shape) 2D et 3D
    - `Line, Circle, Rectangle, Box, Cylinder, ...`
  - Les conteneurs (Layout-Pane) responsable de la disposition des composants enfants
    - hérite de `Pane`
    - `AnchorPane, BorderPane, GridPane, HBox, VBox, ...`
  - Les composants standards (Controls)
    - hérite de la classe `Control`. `Label, Button, TextField, ComboBox`, ...
  - Les composants spécialisés dédiés (lecteur multimédia, navigateur web, etc.)
    - `MediaView, WebView, ImageView, Canvas, Chart`, ...

↪ Conteneurs et agents de placement : Agencement des composants 1/2

Importance de la construction graphique :

- Bonne représentation & compréhension des fonctionnalités
- Guidage de l'utilisateur pour l'accès aux fonctionnalités
- mise à disposition contextuelle, ...

Disposition des composants :

- permet de répondre en grande partie à la problématique précédente
- Ce que l'on entend par disposition :
  - taille des composants
  - position des composants dans la fenêtre et les uns vis-à-vis des autres
  - alignements et espacements
  - bordures et marges
  - comportement dynamique de l'interface
    - redimensionnement de la fenêtre
    - redimensionnement des conteneurs le permettant, etc.
- Deux points de vue :
  - en valeurs absolues (mm,pixel,...)
    - paraît plus simple, mais présente de nombreux inconvénients
    - Exemple : problème des composants de longueur variable, du redimensionnement de la fenêtre, etc.
  - par agents de placement
    - déléguer la disposition des composants à des agents de placement (layout managers)

↪ Conteneurs et agents de placement : Agencement des composants 2/2

Principe des agents de placement :

- définissent des règles de disposition (contraintes) que le gestionnaire respecte en fonction du contexte de la machine cible.
- Contrairement à *Swing*, dans *JavaFX*, les agents sont intégrés aux conteneurs (layout-panes)
  - pas manipulé directement par le programmeur, sauf via ses propriétés.
- agencement des layout-panes les uns avec/dans les autres permettent d'obtenir l'interface voulue
- Plusieurs layout-panes sont déjà présents dans JavaFX.

Propriétés des composants prises en compte par un agent de placement :

- taille minimale et taille préférée :
  - `minWidth` : largeur minimale
  - `prefWidth` : largeur idéale
  - `maxWidth` : largeur maximale
  - `minHeight` : hauteur minimale
  - `prefHeight` : hauteur idéale
  - `maxHeight` : hauteur maximale
- *Remarque :* la prise en compte ou non de ces propriétés dépend du layout-pane dans lequel les composants sont utilisés.

↪ Conteneurs et agents de placement : Fenêtre principale 1/3

Fenêtre principale (primary stage) :

- par défaut affichage au centre de l'écran ;
- méthodes applicables pour modifier (taille, position, ...)
  - `setX()`
  - `setY()`
  - `centerOnScreen()`
  - `setMinWidth()`
  - `setMinHeight()`
  - `setMaxWidth()`
  - `setMaxHeight()`
  - `setResizable()`
  - `sizeToScene()`
- autres méthodes utiles :
  - `setTitle()`
  - `setFullScreen()`
  - `getIcons().add()`
  - `setAlwaysOnTop()`
  - `setScene()`
  - `show()`
  - `showAndWait()`

↪ Conteneurs et agents de placement : Fenêtre principale 2/3

Autres propriétés ajustables sur la fenêtre :

- Bordure :
  - propriété `border` (héritée de `Region`)
  - s'appuie sur l'une des classes `Border` ou `BorderStroke`
    - permet la création d'une bordure assignée à `border`.
    - l'instance de bordure créée est réutilisable
  - respecte le modèle de conception proposée pour les feuilles de style CSS 3 permettant de créer des bordures assez complexes

- Background
  - propriété `background` (héritée de `Region`)
  - s'appuie sur l'une des classes `Background`, `BackgroundFill` et `BackgroundImage`
  - permet la création d'un arrière plan (superposition de remplissage (fill) et/ou images)
  - l'instance de `background` créée est réutilisable

↪ Conteneurs et agents de placement : Fenêtre principale 3/3

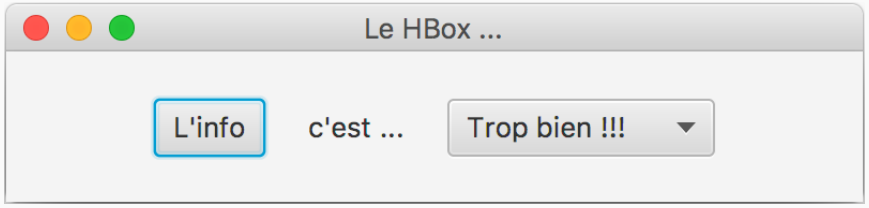
Exemple :

```
1  public class BordEtFond extends Application {
2    private Scene scene;
3    private BorderPane root;
4    private Button btnCoucou;
5    private Image img;
6    @Override
7    public void start(Stage primaryStage) {
8      primaryStage.setTitle("Ma première app JFx !!");
9      primaryStage.setMinHeight(100);
10     primaryStage.setMinWidth(250);
11     root = new BorderPane();
12     btnCoucou = new Button("Coucou les gens !");
13     root.setCenter(btnCoucou);
14     Border bord = new Border(
15     new BorderStroke(Color.CRIMSON,
16     BorderStrokeStyle.DASHED,
17     new CornerRadii(15),
18     new BorderWidths(5),
19     new Insets(15)));
20     root.setBorder(bord);
21     img = new Image("minion.jpeg",50,50,true,false,true);
22     Background bg = new Background(
23     new BackgroundImage(img, BackgroundRepeat.ROUND,
24     BackgroundRepeat.ROUND,
25     BackgroundPosition.CENTER, BackgroundSize.DEFAULT));
26     scene = new Scene(root, 500, 200);
27     root.setBackground(bg);
28     primaryStage.setScene(scene);
29     primaryStage.show();
   }
```

↪ Conteneurs et agents de placement : les Layout-Pane 1/18

`HBox` :

- place les composants sur une ligne horizontale
- les composants sont ajoutés les uns à la suite des autres et de de gauche à droite.
- propriété `alignment` : par défaut `TOP_LEFT`
- l'alignement peut être modifié par la méthode `setAlignment`
- propriété `spacing` : espacement horizontal entre les composants ; peut être passée au constructeur
- propriété `padding` : espacement entre le bord du conteneur et les composants (marge)
  - utilise un paramètre de type `Insets`
- si possible, respecte la taille préférée des composants. Si conteneur trop petit pour afficher tous les composants à leur taille préférée, il les diminue jusqu'à la taille `minWidth`.



↪ Conteneurs et agents de placement : les Layout-Pane 2/18

`HBox` : (suite)

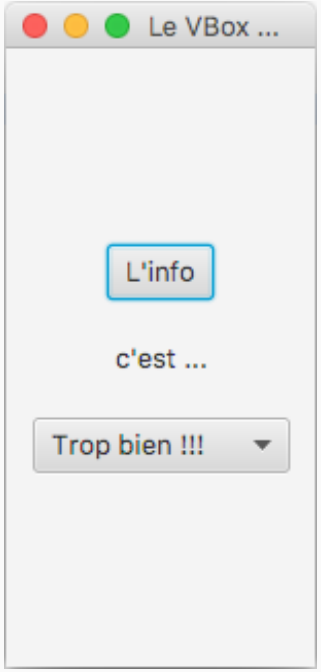
- Ajout des composants enfants :
  - `getChildren()` sur le conteneur : retourne la liste des nodes déjà présents
  - ajout du nouveau composant par l'une des méthodes `add()` ou `addAll())`
- Exemple :

```
1   public class MaHBox extends Application {
2     private Scene scene;
3     private HBox root;
4     private Button btn;
5     private Label lab;
6     private ComboBox<String> cbox;
7     @Override
8     public void start(Stage primaryStage) {
9       primaryStage.setTitle("Le HBox ...");
10      root = new HBox(20);
11      btn = new Button("L'info");
12      lab = new Label("c'est ...");
13      cbox = new ComboBox<>();
14      cbox.getItems().addAll("Trop bien !!!","Trop naze ...");
15      cbox.getSelectionModel().select(0);
16      root.setAlignment(Pos.CENTER);
17      root.getChildren().add(btn);
18      root.getChildren().add(lab);
19      root.getChildren().add(cbox);
20      scene = new Scene(root, 400, 70);
21      primaryStage.setScene(scene);
22      primaryStage.show();
23    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 3/18

`VBox` :

- empile les composants dans le sens vertical
- même fonctionnement global que le `VBox`
  - Attention, remplacer le `h` par un `v` dans certaines méthodes ...



↪ Conteneurs et agents de placement : les Layout-Pane 4/18

`FlowPane` :

- place les composants sur une ligne horizontale ou verticale
- retour à la ligne ou à la colonne suivante (wrapping) si plus assez de place
- orientation fixée lors de l'instanciation (par défaut, horizontal)
- ajout des composants : comme précédemment :
  - appel de `getChildren().add(node)` ou `addAll(n, ...)`
- propriétés du `FlowPane` accessible via les accesseurs :
  - `hgap` et `vgap` : espacement des composants
  - `padding` : marge
  - `alignment` : alignement global dans le conteneur
  - `rowValignment` et `columnHalignment` : alignement dans les lignes et colonnes
  - ...

↪ Conteneurs et agents de placement : les Layout-Pane 5/18

FlowPane : (suite)

```
1   public class MaFlowPane extends Application {
2     private Scene scene;
3     private FlowPane root;
4     private Button btn1,btn2,btn3;
5     private Label lab1, lab2;
6     private TextField txtF;
7
8     @Override
9     public void start(Stage primaryStage) {
10      primaryStage.setTitle("Le FlowPane ...");
11      root = new FlowPane(Orientation.HORIZONTAL);
12      btn1 = new Button("Pique");
13      btn2 = new Button("Douille");
14      btn3 = new Button("toi");
15      lab1 = new Label("Nique");
16      lab2 = new Label("c'est");
17      txtF = new TextField("...");
18      root.setHgap(20);
19      root.setVgap(15);
20      root.setPadding(new Insets(15));
21      root.getChildren().addAll(btn1,lab1,btn2,lab2,btn3,txtF);
22      scene = new Scene(root, 200, 200);
23      primaryStage.setScene(scene);
24      primaryStage.show();
25    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 6/18

`TilePane` :

- place les composants dans une grille homogène
- ajout des composants : comme précédemment :
  - appel de `getChildren().add(node)` ou `addAll(n, ...)`
- ajout se fait soit horizontalement, soit verticalement (mode fixé lors de l'instanciation)
- Toutes les cellules (tuiles) ont la même taille :
  - taille de la plus grande largeur préférée et de la plus grande hauteur préférée des composants placés dans la grille
- propriétés du `TilePane` accessible via les accesseurs :
  - `hgap` et `vgap` : espacement des composants
  - `padding` : marge
  - `alignment` : alignement global dans le conteneur
  - `tileAlignment` : Alignement au sein des colonnes (peut être propre à chacun des composants)
  - `prefColumns` et `prefRows` : nombre préféré de colonnes et de lignes (pas forcement respectées ...)
  - ...

↪ Conteneurs et agents de placement : les Layout-Pane 7/18

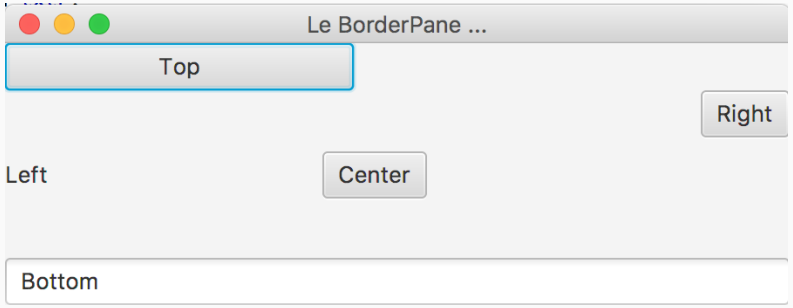
TilePane :(suite)

```
1   public class MaTilePane extends Application {
2     private Scene scene;
3     private TilePane root;
4     private Button btn1,btn2,btn3;
5     private Label lab1, lab2;
6     private TextField txtF;
7
8     @Override
9     public void start(Stage primaryStage) {
10      primaryStage.setTitle("Le TilePane ...");
11      root = new TilePane(Orientation.VERTICAL);
12      root.setAlignment(Pos.CENTER);
13      root.setTileAlignment(Pos.CENTER_LEFT);
14      btn1 = new Button("Pique");
15      btn2 = new Button("Douille");
16      btn3 = new Button("toi");
17      lab1 = new Label("Nique");
18      lab2 = new Label("c'est");
19      TilePane.setAlignment(lab2, Pos.CENTER_RIGHT);
20      txtF = new TextField("...");
21      root.setHgap(20);root.setVgap(15);
22      root.setPadding(new Insets(15));
23      root.getChildren().addAll(btn1,lab1,btn2,lab2,btn3,txtF);
24      scene = new Scene(root, 450, 150);
25      primaryStage.setScene(scene);
26      primaryStage.show();
27    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 8/18

`BorderPane` :

- possibilité de placer les composants dans cinq zones :
  - `TOP`, `BOTTOM`, `LEFT`, `RIGHT` et `CENTER`.
- propriété des différentes zones :
  - `TOP` et `BOTTOM` :
    - Gardent leur hauteur préférée
    - éventuellement agrandis ou réduits horizontalement en fonction de la largeur du conteneur. (C'est la taille max du composant qui est pris en compte)
  - `LEFT` et `RIGHT` :
    - Gardent leur largeur préférée
    - éventuellement agrandis ou réduits verticalement en fonction de la hauteur restante entre les deux zones précédentes (C'est la taille max du composant qui est pris en compte)
  - `CENTER` :
    - éventuellement agrandi ou réduit horizontalement et verticalement pour occuper l'espace restant au centre du conteneur (Ce sont les tailles max et min du composant qui sont pris en compte)



↪ Conteneurs et agents de placement : les Layout-Pane 9/18

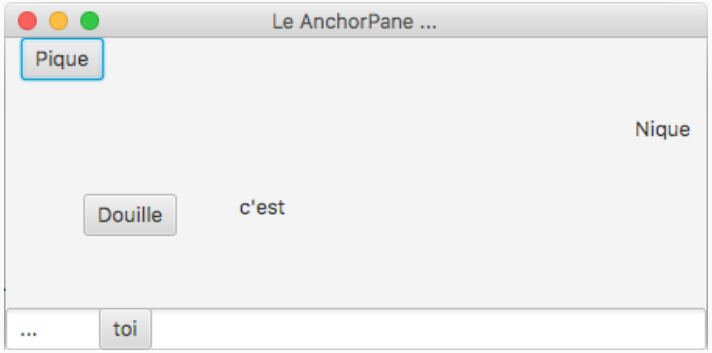
BorderPane : (suite)

```
1   public class MaBorderPane extends Application {
2     private Scene scene;
3     private BorderPane root;
4     private Button btn1,btn2,btn3;
5     private Label lab1;
6     private TextField txtF;
7
8     @Override
9     public void start(Stage primaryStage) {
10      primaryStage.setTitle("Le BorderPane ...");
11      root = new BorderPane();
12      btn1 = new Button("Top");
13      btn2 = new Button("Right");
14      btn3 = new Button("Center");
15      lab1 = new Label("Left");
16      txtF = new TextField("Bottom");
17      btn1.setMaxWidth(200);
18      root.setTop(btn1);
19      root.setLeft(lab1);
20      BorderPane.setAlignment(lab1, Pos.CENTER);
21      root.setRight(btn2);
22      root.setBottom(txtF);
23      root.setCenter(btn3);
24      scene = new Scene(root, 450, 150);
25      primaryStage.setScene(scene);
26      primaryStage.show();
27    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 10/18

`AnchorPane` :

- Ajout des composants en leur associant des contraintes d'ancrage
  - un composant peut être ancré à plusieurs côtés
  - plusieurs composants peuvent être ancrés à un même côté.
  - `AnchorPane` respecte, si possible, la taille préférée des composants
  - si un composant est ancré à plusieurs bords (même opposés)ce dernier pourra être agrandi ou diminué pour respecter les contraintes d'ancrage.
  - si plusieurs composants sont ancrés à un même bord, ils pourront être partiellement ou totalement superposés (l'ordre d'ajout détermine la superposition).
- l'ajout se fait en deux étapes :
  - ajout classique au conteneur par appel de l'une des méthodes `getChildren().add(node)` ou `addAll(n, ...)`
  - définition des contraintes par méthodes statiques issues de la classe `AnchorPane`
    - `topAnchor()` : distance par rapport au haut
    - `rightAnchor()` : distance par rapport au bord droit
    - `bottomAnchor()` : distance par rapport au bas
    - `leftAnchor()` : distance (double) par rapport au bord gauche



↪ Conteneurs et agents de placement : les Layout-Pane 11/18

AnchorPane : (suite)

```
1   ...
2   @Override
3   public void start(Stage primaryStage) {
4     primaryStage.setTitle("Le AnchorPane ...");
5     root = new AnchorPane();
6     btn1 = new Button("Pique");
7     btn2 = new Button("Douille");
8     btn3 = new Button("toi");
9     lab1 = new Label("Nique");
10    lab2 = new Label("c'est");
11    txtF = new TextField("...");
12    root.getChildren().addAll(btn1,lab1,btn2,lab2,txtF,btn3);
13    AnchorPane.setLeftAnchor(btn1, 10d);
14    AnchorPane.setRightAnchor(lab1, 10d);
15    AnchorPane.setTopAnchor(lab1, 50d);
16    AnchorPane.setTopAnchor(btn2, 100d);
17    AnchorPane.setLeftAnchor(btn2, 50d);
18    AnchorPane.setTopAnchor(lab2, 100d);
19    AnchorPane.setLeftAnchor(lab2, AnchorPane.getLeftAnchor(btn2)+100d);
20    AnchorPane.setBottomAnchor(txtF, 0d);
21    AnchorPane.setRightAnchor(txtF, 0d);
22    AnchorPane.setLeftAnchor(txtF, 0d);
23    AnchorPane.setBottomAnchor(btn3, 0d);
24    AnchorPane.setLeftAnchor(btn3, 60d);
25    scene = new Scene(root, 450, 200);
26    primaryStage.setScene(scene);
27    primaryStage.show();
28  }
```

↪ Conteneurs et agents de placement : les Layout-Pane 12/18

`StackPane` :

- empile les composants les uns au dessus des autres dans l'ordre d'ajout
- redimensionnement des composants pour s'adapter à la taille du conteneur
  - respecte la `maxSize` mais pas la `minSize`.
- alignement en fonction de la valeur `alignment` du conteneur (`CENTER` par défaut)
- modification de l'ordres de composants :
  - suppression et insertion du composant à l'index voulu
  - utilisation des méthodes `toBack()` et `toFront()` sur les nodes

↪ Conteneurs et agents de placement : les Layout-Pane 13/18

StackPane : (suite)

```
1   public class MaStackPane extends Application {
2     private Scene scene;
3     private StackPane root;
4     private Button btn1,btn2,btn3;
5     private Label lab1, lab2;
6     private TextField txtF;
7     private Image img;
8     private ImageView imgV;
9     @Override
10    public void start(Stage primaryStage) {
11      primaryStage.setTitle("Le TilePane ...");
12      root = new StackPane();
13      root.setAlignment(Pos.CENTER);
14      img = new Image("minion.jpeg",100,100,true,false,true);
15      imgV = new ImageView(img);
16      btn1 = new Button("Le bouton ...");
17      btn2 = new Button();
18      btn3 = new Button("toi");
19      lab1 = new Label("Nique");
20      lab2 = new Label("c'est");
21      txtF = new TextField("...");
22      root.setPadding(new Insets(15));
23      root.getChildren().addAll(imgV,btn1,lab1,btn2,lab2,btn3,txtF);
24      btn1.toFront();
25      scene = new Scene(root, 450, 150);
26      primaryStage.setScene(scene);
27      primaryStage.show();
28      root.getChildren().remove(imgV);
29      root.getChildren().add(6,imgV);
30    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 14/18

`GridPane` :

- dispose les composants enfants dans une grille flexible (lignes et colonnes)
  - comme une table HTML.
- la grille peut être irrégulière
- la zone occupée par un composant peut s'étendre (`span`) sur plusieurs lignes et/ou colonnes
- nombre de lignes et colonnes déterminé automatiquement par les zones où sont ajoutés les composants.
- par défaut, hauteur de chaque ligne fixée par la hauteur préférée du composant le plus haut
- par défaut, largeur de chaque colonne fixée par la largeur préférée du composant le plus large
- ajout des composants (l'orde n'a pas d'impact) :
  - appel de la méthode `add()` avec en paramètres :
    - indice de la colonne
    - indice de ligne
    - nombre de colonnes de spanning (1 par défaut)
    - nombre de lignes de spanning (1 par défaut)
- possibilité d'ajouter plusieurs composants dans une même cellule
  - même comportement que le conteneur `StackPane`

↪ Conteneurs et agents de placement : les Layout-Pane 15/18

`GridPane` : (suite)

- modification de la position :
  - modifier les propriétés `columnIndex` et `rowIndex`
  - modifier les propriétés `columnSpan` et `rowSpan`
- propriétés globales :
  - `hgap, vgap, alignment, padding, gridLinesVisible`
- contraintes globales :
  - définies dans des objets de types `RowConstraints` et `ColumnConstraints`
  - ajoutées à la grille par `getRowConstraints.add(...)` et `getColumnConstraints.add(...)`
- contraintes individuelles :
  - `halignment, valignement, hgrow, vgrow, margin`
- les contraintes individuelles sont prioritaires sur les contraintes globales

↪ Conteneurs et agents de placement : les Layout-Pane 16/18

GridPane : (suite)

```
1   public class MaGridPane extends Application {
2     private Scene scene;
3     private GridPane root;
4     private Button btn1;
5     private Label lab1;
6     private TextField txtF;
7     private Image img;
8     private ImageView imgV;
9     @Override
10    public void start(Stage primaryStage) {
11      primaryStage.setTitle("Le GridPane ...");
12      root = new GridPane();
13      root.setAlignment(Pos.CENTER);
14      img = new Image("minion.jpeg",100,100,true,false,true);
15      imgV = new ImageView(img);
16      btn1 = new Button("Pique ...");
17      lab1 = new Label("Nique");
18      txtF = new TextField("Douille ...");
19      root.add(imgV, 0,0,1,3);
20      root.add(btn1, 1, 0);
21      root.add(lab1, 1, 1);
22      root.add(txtF, 1, 2);
23      root.setHgap(10d);
24      root.setVgap(5);
25      GridPane.setValignment(lab1,VPos.BOTTOM);
26      scene = new Scene(root, 450, 150);
27      primaryStage.setScene(scene);
28      primaryStage.show();
29    }
```

↪ Conteneurs et agents de placement : les Layout-Pane 17/18

En résumé :

Layout-Pane	Propriétés
`HBox/VBox`	Place les composants les horizontalement/verticalement sur une ligne/colonne
`FlowPane`	Place les composants sur une ligne/colonne ; passe à la ligne/colonne suivante si pas assez d'espace dans le conteneur
`TilePane`	Place les composants dans une grille ; les cellules de la grille sont toutes de même taille ; les composants sont ajoutés horizontalement/verticalement (i.e. par ligne ou par colonne)
`BorderPane`	Permet de disposer les composants dans 5 emplacements : `Top`, `Bottom`, `Left`, `Right`, `Center`
`AnchorPane`	Dispose les composants en affectant une/des contrainte(s) de distance par rapport à un/aux bord(s) du conteneur
`StackPane`	Dispose les composants les uns par dessus les autres ; le dernier ajouté est le composant visible
`GridPane`	Ajoute (via l'indice de ligne et de colonne) les composants dans une grille irrégulière dont la taille des lignes et des colonnes correspond au plus grand composant contenu ; un composant peut s'étendre sur plusieurs cellules ; le composant peut s'agrandir pour occuper toute la zone dans laquelle il a été placée

↪ Conteneurs et agents de placement : les Layout-Pane 18/18

Suppression & remplacement d'un composant :

- suppression : appel de la méthode `remove(node)` ou `removeAll(node1, node2, ...)` sur la liste des composants
- remplacement : suppression puis ajout à l'index désiré
- toute modification dynamique du graphe de scène devrait être suivi d'un appel à la méthode `sizeToScene()` ; force la reconstruction de la fenêtre principale

↪ Programmation événementielle

- *Programmation impérative VS événementielle :*
  - programmation impérative : le programme dirige les opérations (demande des valeurs, puis calcule et enfin affiche un résultat)
    L'utilisateur est au service du programme.
  - programmation événementielle : les événements dirige l'exécution de l'application. Ce type de programmation s'adapte bien à la réalisation d'interfaces graphiques (interaction de l'utilisateur possible à tout moment).
    Le programme via son interface est au service de l'utilisateur.

Event Loop Thread :

- la programmation événementielle nécessite l'exécution d'une tâche gérant l'arrivée des événements ; c'est l'Event Loop Thread.
- Ce dernier nous prévient et déclenche l'action associée à l'apparition de cet événement est exécuté par l'Event Loop Thread.
- Attention, pendant la durée d'exécution de l'action, l'Event Loop Thread est ≪ sourd ≫ à tout autre arrivée d'événements ...

↪ Les événements 1/11

Définition :

- un événement (`event`) est une notification indiquant un changement pouvant être provoqué par :
  - une interaction de l'utilisateur (clic sur un bouton, ...)
  - un changement en provenance du système (changement de valeur d'une propriété, ...)

Événement en JavaFx :

- modélisé par des instances de la classe `Event`, classe mère de nombreux événements prédéfinis :
  - `MouseEvent, KeyEvent, DragEvent, ScrollEvent, ...`
- toute instance de ce type contient au minimum :
  - le type de l'événement : `EventType`
  - la source de l'événement : `Object`
  - la cible de l'événement : `EventTarget`

↪ Les événements 2/11

Propagation des événements :

- un événement se propage d'abord vers le bas : depuis la racine (`Stage`) jusqu'à la cible (`Target`), exemple un bouton cliqué par l'utilisateur
  - les éventuels ≪ filtres ≫ (`Event Filter`) enregistrés sont pris en compte : la méthode associée au filtre est exécutée
- puis vers le haut, depuis la cible jusqu'à la racine
  - les éventuels gestionnaires d'événements (`Event Listener`) enregistrés sont pris en compte : la méthode associée au gestionnaire est exécutée
- filtres et gestionnaires sont appliqués dans l'ordre de passage

↪ Les événements 3/11

Écouteur d'événement :

- pour permettre l'exécution d'une méthode en réponse à un événement il faut :
  - créer un écouteur d'événement (`Event Listener`)
  - enregistrer cet écouteur sur un/des nœud(s) du graphe où l'on souhaite effectuer un traitement.
    - un écouteur d'événement peut être enregistré auprès d'un composant comme filtre ou comme gestionnaire.
    - un filtre (`Filter`) sera exécuté lors de la propagation descendante de l'événement
    - un gestionnaire (`Handler`) sera exécuté lors de la propagation ascendante de l'événement

Création d'un écouteur d'événement (`Filter` ou `Handler`) :

- instance d'une classe devant implémenter l'interface fonctionnelle générique `EventHandler<T extends Event>`
  - impose la redéfinition de l'unique méthode `handle(T event)`
  - c'est dans cette méthode que le traitement de l'événement est implémenté

↪ Les événements 4/11

Enregistrement d'un écouteur (filtre ou gestionnaire) à un composant :

- Filtre :
  - méthode `addEventFilter()` présente dans toutes les classes héritant de `Node`
- Gestionnaire :
  - méthode `addEventHandler(Event,Handler)` présente dans toutes les classes héritant de `Node`
    - `Event` permet d'indiquer le type d'événement auquel on souhaite réagir
    - `Handler` permet de passer une instance de l'écouteur
  - pour certains composants, possibilité d'utiliser une des méthodes particulières permettant d'enregistrer un gestionnaire d'événement en tant que propriété
  - formater comme tel : `setOnEventType(Handler)`

↪ Les événements 5/11

Typage d'un écouteur :

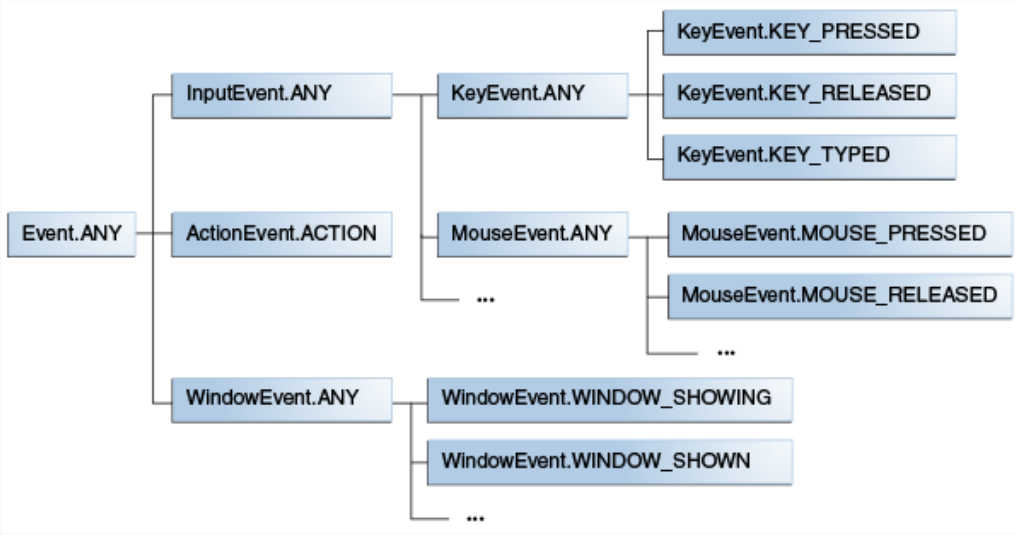
- l'écouteur se base sur une structure générique : `EventHandler<T extends Event>`
- le typage de l'écouteur se fait en choisissant le bon type d'événement à passer en paramètre générique
- un événement possède un type (`EventType`) et possède :
  - un nom que l'on peut obtenir par (`getName()`)
  - un type parent obtenable par (`getSuperType()`).
- les événements sont construits de manière hiérarchique par héritage
  - pour un clic de la souris, l'événement est `MouseEvent.MOUSE_CLICKED` et le type parent est `MouseEvent.ANY`.

↪ Les événements 6/11

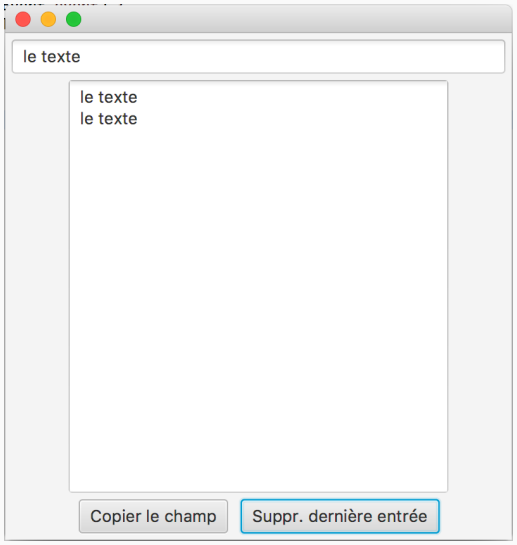
Stopper la propagation d'un événement :

- un ou plusieurs filtre(s) et/ou gestionnaire(s) d'événements peut/peuvent être associé(s) sur chacun des composants du graphe.
- ces filtres et/ou gestionnaires exécutent le traitement associé et propage l'événement au composant suivant
- cette propagation peut-être interrompu en consommant l'événement :
  - appel de la méthode `consume()`
- Si tel est le cas, la propagation de l'événement s'interrompt et les écouteurs suivant dans la chaîne de propagation ne seront donc plus activés

Hiérarchie de la classe `Event` :



↪ Les événements 7/11



Exemple :

- Copie d'un champ texte dans une zone de texte si clique sur bouton (sauf si le champ texte est vide)
- Suppression d'une ligne de la zone de texte si clique sur bouton
- réinitialisation du champ texte si touche tabulation

  Solution proposée :
  - 1 gestionnaire sur le bouton ≪ Copier le champ ≫
    - classe écouteur en utilisant un événement de type `ActionEvent.ACTION`
  - 1 gestionnaire sur le bouton ≪ Suppr. dernière entrée ≫
    - classe écouteur en utilisant un événement de type `MouseEvent.MOUSE_CLICKED`
  - 1 filtre (sur `root`) qui consomme l'événement du clic sur le premier bouton si champ texte vide
    - classe écouteur en utilisant un événement de type `ActionEvent.ACTION`
  - 1 filtre (sur `root`) qui régait à l'enfoncement de la touche *TAB* et qui consomme l'événement ensuite
    - classe anonyme en utilisant un événement de type `KeyEvent.PRESSED`

↪ Les événements 8/11

Code de la classe écouteur pour la copie :

```
1   public class EHandCopie implements EventHandler<ActionEvent> {
2     private IhmQuiRepond vue;
3     public EHandCopie(final IhmQuiRepond v) {
4       vue = v;
5     }
6     @Override
7     public void handle(ActionEvent event) {
8       vue.ajoutTextDansArea(vue.getTextFieldVal());
9     }
10  }
```

Code de la classe écouteur pour la suppression :

```
1   public class EHandSuppr implements EventHandler<MouseEvent>{
2     private IhmQuiRepond vue;
3     public EHandSuppr(final IhmQuiRepond v) {
4       vue = v;
5     }
6     @Override
7     public void handle(MouseEvent event) {
8       vue.supprTextDansArea();
9     }
10  }
```

↪ Les événements 9/11

Code de la classe écouteur pour éviter la copie si le champ texte est vide :

```
1   public class EFilCopie implements EventHandler<ActionEvent>{
2     private IhmQuiRepond vue;
3     public EFilCopie(IhmQuiRepond v) {
4       vue = v;
5     }
6     @Override
7     public void handle(ActionEvent event) {
8       if(vue.getTextFieldVal().compareTo("")==0){
9         event.consume();
10      }
11    }
12  }
```

Code de la classe principale : construction graphique (début)

```
1   public class IhmQuiRepond extends Application {
2     private FlowPane panBouton;
3     private Button btnCopy, BtnDel;
4     private TextField txtF;
5     private TextArea txtA;
6     @Override
7     public void start(Stage primaryStage) {
8       panBouton = new FlowPane();
9       panBouton.setAlignment(Pos.CENTER);
10      panBouton.setHgap(10d);
11      panBouton.setPadding(new Insets(5));
12      btnCopy = new Button("Copier le champ");
13      BtnDel = new Button("Suppr. dernière entrée");
14      panBouton.getChildren().addAll(btnCopy,BtnDel);
15      ...
```

↪ Les événements 10/11

Code de la classe principale : construction graphique (suite)

```
1     txtF = new TextField();
2     txtA = new TextArea();
3     txtA.setMaxWidth(300d);
4     txtA.setEditable(false);
5     BorderPane root = new BorderPane();
6     root.setBottom(panBouton);
7     root.setTop(txtF);
8     root.setCenter(txtA);
9     BorderPane.setMargin(txtF, new Insets(5d));
10    BorderPane.setAlignment(panBouton, Pos.CENTER);
11    ...
```

Code de la classe principale : abonnement

```
1     EHandCopie h1 = new EHandCopie(this);
2     btnCopy.addEventHandler(ActionEvent.ACTION, h1);
3     EHandSuppr hsupr = new EHandSuppr(this);
4     BtnDel.setOnMouseClicked(hsupr);
5     root.addEventFilter(ActionEvent.ANY, new EFilCopie(this));
6     root.addEventFilter(KeyEvent.KEY_PRESSED,
7     new EventHandler<KeyEvent>() {
8       @Override
9       public void handle(KeyEvent event) {
10        if(event.getCode() == KeyCode.TAB){
11          resetTextFieldVal();
12          event.consume();
13        }
14      }});
15    ...
```

↪ Les événements 11/11

Suite de la classe :

```
1    Scene scene = new Scene(root, 400, 400);
2    primaryStage.setScene(scene);
3    primaryStage.show();
4  }
5  public String getTextFieldVal(){
6    return txtF.getText();
7  }
8  public void resetTextFieldVal(){
9    txtF.setText("");
10 }
11 public void ajoutTextDansArea(String txt){
12   txtA.appendText(txt+"\n");
13 }
14 public void supprTextDansArea(){
15   ArrayList<String> tab = new ArrayList<String>(Arrays.asList(txtA.getText().split("\n")));
16   txtA.setText("");
17   for (int i=0; i<tab.size()-1;i++){
18     txtA.appendText(tab.get(i)+"\n");
19   }
20 }
21 public static void main(String[] args) {
22   launch(args);
23 }
24 }
```