

Développement d'applications avec IHM (JavaFX)

Introduction - Contrôles, Conteneurs, Événements

Petru Valicov
`petru.valicov@umontpellier.fr`

`https://gitlabinfo.iutmontp.univ-montp2.fr/ihm`

2021-2022



Java *vs* JavaFX

Java est une plateforme

- langage de programmation orienté objet de haut niveau
- moteur d'exécution (JVM)
- interface de programmation d'application standardisée (API)

JavaFX est un framework Java pour développer des IHM

- successeur de Swing
- créé et développé par Sun → Oracle → OpenJFX
- le cycle des versions JavaFX est calqué sur celui de Java
- caractéristiques :
 - une riche librairie de composants
 - possibilité de décrire l'interface dans un format simplifié (XML)
 - génération des IHM via un outil interactif (Scene Builder)
 - une séparation claire entre la vue utilisateur et le code métier

Lancement d'une fenêtre

```
import javafx.application.Application;
import javafx.stage.Stage;

public class SalutLeMonde extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Salut le monde !");
        primaryStage.show();
    }
}
```

- la classe principale est toujours une `Application`
- la méthode `start(Stage primaryStage)` est automatiquement appelée par le runtime JavaFX :
 1. l'objet de type `Application` est instancié par l'environnement
 2. l'environnement `JavaFX` fournit une fenêtre par défaut (type `Stage`) à la méthode `start(Stage primaryStage)`

Lancement d'une fenêtre avec `main(String[] args)`

```
import javafx.application.Application;
import javafx.stage.Stage;

public class SalutLeMonde extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Salut le monde !");
        primaryStage.show();
    }

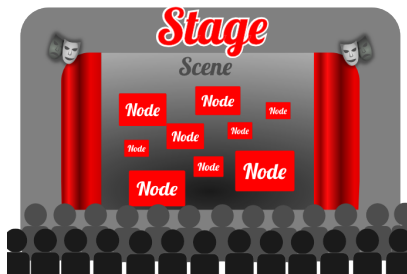
    public static void main(String[] args) {
        Application.launch(args); // méthode qui exécutera start
    }
}
```

- JavaFX détermine automatiquement le bon type effectif de `Application` à utiliser pour invoquer `start(...)`
- utiliser `main(String[] args)` permet de passer des paramètres à la fenêtre au lancement

Chez JavaFX c'est comme au théâtre !

Les éléments de base d'une application JavaFX font référence à une salle de spectacle :

- La fenêtre c'est l'estrade (type `Stage`)
- Sur une estrade on joue une scène de spectacle (type `Scene`)
- Les éléments de la scène sont des objets de type `Node`

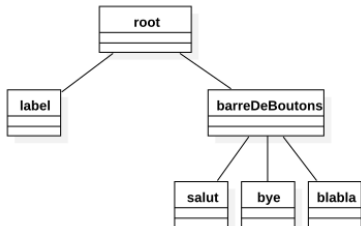


Le graphe de scène

Les éléments d'une scène sont organisés sous forme d'un arbre :

- un objet de type Node est désignée comme la **racine**
- des objets Node **fils intermédiaires** (ont des fils)
des conteneurs regroupant plusieurs composants
- des objets Node **feuilles** (n'ont pas de fils)
boutons, champs de saisie ou texte, formes graphiques etc

```
public void start(Stage stage){  
    BorderPane root = new BorderPane();  
    Label label = new Label("Salut le monde !");  
    HBox bareDeBoutons = new HBox();  
    Button salut = new Button("Salut");  
    Button bye = new Button("Bye");  
    Button blabla = new Button("Raconte une histoire");  
    bareDeBoutons.getChildren().addAll(salut, bye, blabla);  
    root.setLeft(label);  
    root.setRight(bareDeBoutons);  
    Scene scene = new Scene(root, 420, 100);  
    stage.setTitle("Ma fenetre!");  
    stage.setScene(scene);  
    stage.show();  
}
```



La classe Node et ses sous-classes

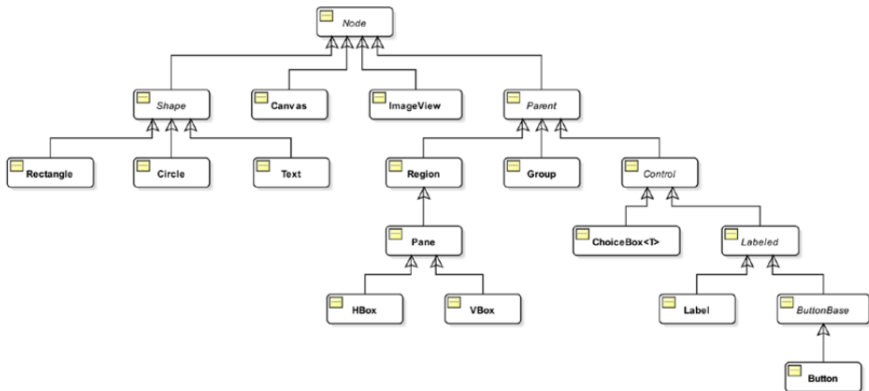


Diagramme de classes partiel de la hiérarchie **Node**

Conteneurs

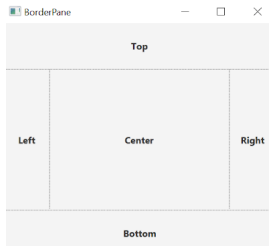
Permettent d'indiquer l'organisation des composants :

- HBox
 - placement des composants sur une ligne horizontale, de gauche à droite
 - des fonctions permettent d'adapter le conteneur :
setAlignment(), setMinWidth(), setSpacing(), etc.
- VBox
 - idem que HBox mais en vertical

```
public void start(Stage stage){  
    Label label = new Label("Salut le monde !");  
    Button bye = new Button("Bye");  
  
    VBox vBox = new VBox();  
    vBox.getChildren().addAll(label, bye); //ajout des noeuds à la suite  
}
```


Conteneurs - BorderPane

- 5 zones : Haut, Bas, Gauche, Centre, Droite
- un seul nœud par zone
- le nœud du centre aura la tendance d'occuper le plus de place
- découpage classique d'une fenêtre (par exemple celle de votre IDE !)



```
BorderPane root = new BorderPane();  
root.setLeft(new Button("Left"));  
root.setRight(new Button("Right"));  
root.setTop(new Button("Top"));  
root.setBottom(new Button("Bottom"));  
root.setCenter(new Button("Center"));  
Scene scene = new Scene(root); // ajout du conteneur comme racine
```

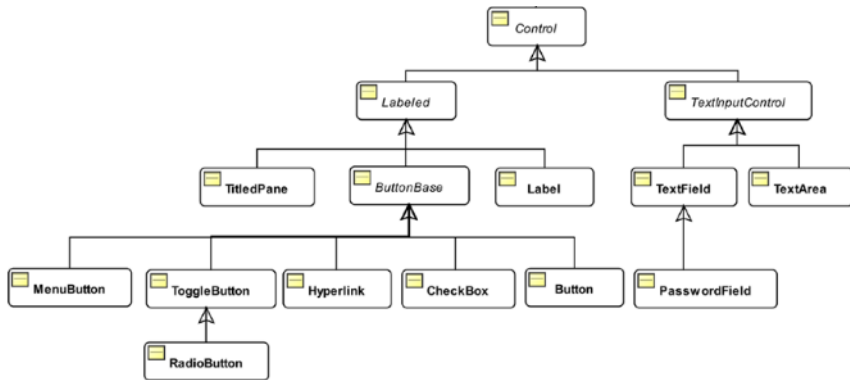
Conteneurs

Permettent d'indiquer l'organisation des composants :

- **TilePane**
 - placement sous forme d'une grille
 - toutes les cases de la grille ont la même taille
- **GridPane**
 - placement sous forme d'une grille
 - les lignes/colonnes peuvent être de taille variable
- **FlowPane**
 - les éléments sont disposés sur une ligne (horizontale ou verticale)
 - lorsque il n'y a plus assez de place disponible, on passe à la ligne suivante
- **StackPane**
 - les composants sont organisés sous forme d'une pile (seul le sommet de la pile est visible)
 - exemple : une pile de cartes dans un jeu

Composants graphiques (visibles)

Tous les composants interagissant directement avec l'utilisateur héritent de la classe abstraite `javafx.controls.Control`.



Hiérarchie des classes Control.

Quelques composants de base - Label

- un simple étiquette affichée (texte, icône), non-éditable
- pas de traitement associé intéressant prévu

```
BorderPane root = new BorderPane();

Label etiquette = new Label("Je suis un label textuel");
etiquette.setFont(Font.font("Cambria", 18));
etiquette.setTextFill(Color.DARKCYAN);
root.setTop(etiquette);

Image image = new Image("figures/JavaFXLogo.png");
// possible d'indiquer l'URL de l'image directement
// Image image = new Image("https://URL-de-l-image");
Label labelAvecImage = new Label();
labelAvecImage.setGraphic(new ImageView(image));
root.setCenter(labelAvecImage);

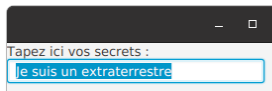
// taille de la scène en fonction de l'image
double largeur = image.getWidth()+50;
double longueur = image.getHeight()+50;
Scene scene = new Scene(root, largeur, longueur);
primaryStage.setScene(scene);
primaryStage.show();
```



Quelques composants de base - TextField

- permet de créer un champ de saisie (une ligne)
- possibilité d'associer un traitement (par ex. en fonction du texte saisi)

```
public void start(Stage primaryStage) {  
    VBox root = new VBox();  
  
    Label message = new Label("Tapez ici vos secrets :");  
  
    TextField champ = new TextField();  
    // personnalisation du champ de texte  
    champ.setMaxWidth(260);  
    champ.setText("Je suis un extraterrestre");  
  
    // ajout des 2 noeuds au conteneur  
    root.getChildren().addAll(message, champ);  
    // affichage de la scène  
    Scene scene = new Scene(root, 300, 50);  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```

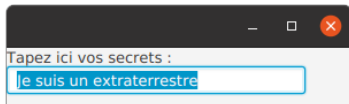


Quelques composants de base - TextField

- Associer un traitement est souvent ce qui est le plus intéressant
- Plusieurs façons de faire :
 - à travers une classe interne anonyme
 - en utilisant une expression lambda (à privilégier)

```
Label message = new Label("Tapez ici vos secrets :");
TextField champ = new TextField("Je suis un extraterrestre");

// Traitement exécuté lorsque l'utilisateur tape "Entrée"
champ.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent actionEvent) {
        champ.getText();
        message.setText("On a un E.T. ici !");
    }
});
```



```
Label message = new Label("Tapez ici vos secrets :");
TextField champ = new TextField("Je suis un extraterrestre");

// Traitement exécuté lorsque l'utilisateur tape "Entrée"
champ.setOnAction(actionEvent -> {
    champ.getText();
    message.setText("On a un E.T. ici !");
});
```



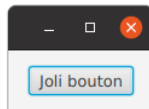
Quelques composants de base - Button

Trois types de boutons :

- exécution de commandes (Button, Hyperlink et MenuButton)
- pour faire des choix (ToggleButton, CheckBox et RadioButton)
- combinaison des deux (SplitMenuButton)

Tous les boutons permettent un traitement lors du clic

```
public void start(Stage primaryStage) {  
    Button bouton = new Button("Joli bouton");  
  
    //le clic sur le bouton provoque un traitement (ici affichage) :  
    bouton.setOnAction(e -> System.out.println("clic intercepté"));  
  
    BorderPane root = new BorderPane(bouton);  
    Scene scene = new Scene(root, 120, 50);  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```



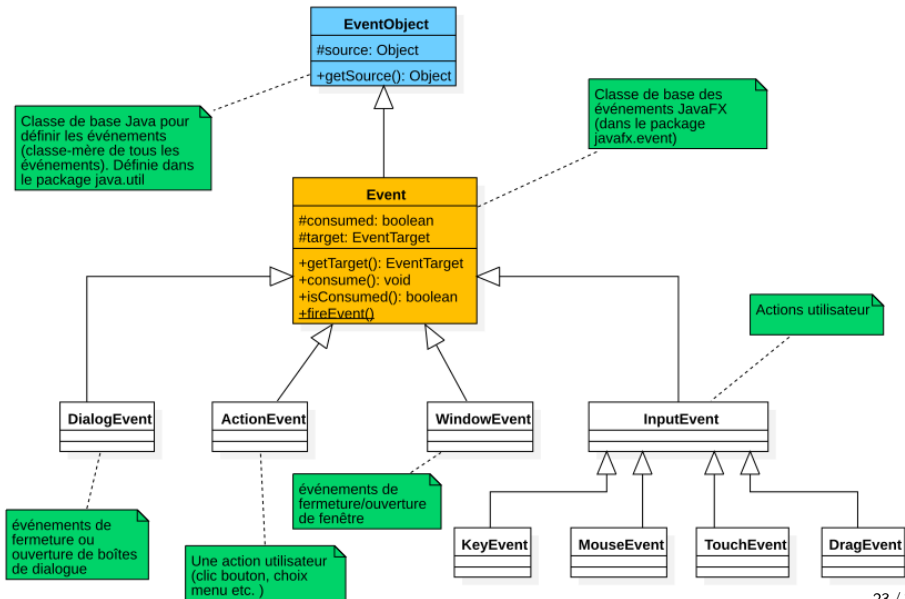
Introduction à la programmation événementielle

Définition - Événement

Une action qui peut être identifiée par un programme informatique et qui peut être "*gérée*" par le programme grâce à un **gestionnaire d'événements** (EventHandler en Java).

- pour intercepter l'occurrence d'un événement, un système d'écoute est mis en place (*listener*)
- dans les IHM un événement correspond à une action élémentaire d'un utilisateur sur la GUI
- exemples : clic souris, taper une touche au clavier, changement d'heure etc.

Hiérarchie (partielle) des classes d'événements JavaFX



Les événements en JavaFX

Tous les **Event** de JavaFX ont 3 propriétés :

1. Une *source* - un **Object** Java qui a généré l'événement (suite à un changement d'état)
2. Une *destination* (**EventTarget**) - typiquement un élément du graphe de scène ou la fenêtre principale
3. Un *type d'événement* - spécifie l'événement concret sous-jacent

Principe de fonctionnement :

- Lorsqu'un événement a lieu, souvent on souhaite définir une réponse de réaction sur l'interface graphique.
- Pour cela il faut enregistrer un gestionnaire d'événement (type **EventHandler**) sur l'élément *source*.
- Lorsque l'événement est détecté par l'environnement Java, celui-ci exécutera le code correspondant défini dans l'objet **EventHandler** correspondant.

```
// une interface fonctionnelle (une et une seule méthode abstraite)
public interface EventHandler<T extends Event> extends EventListener {
    void handle(T var1);
}
```

- la méthode `handle(T var1)` reçoit un événement JavaFX
- l'intégralité du code de réaction au déclenchement de l'événement y est définie

Exemple d'ajout d'un gestionnaire (version explicite et longue)

```
public class ExempleMouseEvent extends Application {

    @Override
    public void start(Stage primaryStage) {
        Scene scene = new Scene(new Group(), 200, 200);

        // ajout de l'écouteur à la scène
        scene.addEventHandler(MouseEvent.MOUSE_CLICKED,
            new GestionnaireSouris());

        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

```
// Définition d'un gestionnaire avec un traitement
public class GestionnaireSouris implements
    EventHandler<MouseEvent> {

    @Override
    public void handle(MouseEvent mouseEvent) {
        System.out.println("événement de souris");
    }
}
```

Utilisation des *lambdas* pour l'enregistrement

Rappel - interface fonctionnelle

Une interface Java qui n'a qu'une et une seule méthode abstraite.

Comme `EventHandler` est une interface fonctionnelle, on peut utiliser des lambdas !

```
public void start(Stage primaryStage) {  
    Scene scene = new Scene(new Group(), 200, 200);  
  
    // création d'un événement sous-forme d'expression lambda  
    EventHandler<MouseEvent> mouseEventHandler =  
        evenement -> System.out.println("événement de souris");  
  
    // ajout du gestionnaire d'événement à la scène  
    scene.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEventHandler);  
  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```

Remarques sur l'enregistrement

Les handlers (gestionnaires) peuvent être enregistrés sur tous les objets ayant la méthode `addEventHandler(...)` :

```
// signature de la méthode d'enregistrement d'un gestionnaire d'événement
<T extends Event> void addEventHandler(EventType<T> type,
                                       EventHandler<? super T> gestionnaire)
```

- Le type générique `T` désigne le type d'événement (par ex. `MouseEvent`)
- Le paramètre `type` de la fonction permet de préciser le type concret d'événements :
 - pour `MouseEvent` : `MOUSE_PRESSED`, `MOUSE_RELEASED`, `MOUSE_CLICKED`, `MOUSE_MOVED`, etc.
 - pour `KeyEvent` : `KEY_PRESSED`, `KEY_RELEASED`, `KEY_RELEASED`, etc.

Les classes `Node`, `Scene`, `Window` ont une définition de la fonction `addEventHandler(...)`

Il est possible d'ajouter plusieurs *handlers* sur le même nœud :

```
// du code d'initialisation

Button btn = new Button("Hello");
Rectangle rectangle = new Rectangle(80, 120);
rectangle.setFill(Color.RED);

btn.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {
    btn.setText("click");
});

btn.addEventHandler(MouseEvent.MOUSE_MOVED, mouseEvent -> {
    if (rectangle.getFill() == Color.RED)
        rectangle.setFill(Color.GREEN);
    else
        rectangle.setFill(Color.RED);
});

// du code de construction de scène
```

Les sources et les cibles des événements

- La méthode `getSource()` de `Event` renvoie la source de l'événement
 - le nœud sur lequel `addEventHandler(...)` a été invoquée
- La méthode `getTarget()` de `Event` renvoie la cible de l'événement
 - le nœud sur le graphe de scène sur lequel l'événement va agir
- en JavaFX souvent la source est la cible sont les mêmes, mais pas toujours !

Les sources et les cibles des événements

```
public void start(Stage stage) throws Exception {
    Circle cercle = new Circle(25, 25, 25);
    HBox racine = new HBox();
    racine.getChildren().add(cercle);
    Scene scene = new Scene(racine);

    EventHandler<MouseEvent> gestionnaire = mouseEvent -> {
        System.out.println("Source :" + mouseEvent.getSource().getClass().getSimpleName());
        System.out.println("Cible :" + mouseEvent.getTarget().getClass().getSimpleName());
    };

    // enregistrement du même gestionnaire sur plusieurs sources
    scene.addEventHandler(MouseEvent.MOUSE_CLICKED, gestionnaire);
    racine.addEventHandler(MouseEvent.MOUSE_CLICKED, gestionnaire);
    cercle.addEventHandler(MouseEvent.MOUSE_CLICKED, gestionnaire);

    stage.setScene(scene);
    stage.show();
}
```



Un clic de souris sur le cercle provoque l'affichage suivant :

```
Source :Circle
Cible :Circle
Source :HBox
Cible :Circle
Source :Scene
Cible :Circle
```

La cible reste systématiquement la même (ici l'objet cercle).

Les événements en JavaFX

Quelques remarques :

- les lambdas sont très pratiques, mais à éviter lorsque le même gestionnaire est à utiliser sur plusieurs éléments de la GUI
- Pour supprimer un gestionnaire :

```
// création d'un événement
Bouton bouton = new Bouton("Coucou");
EventHandler<MouseEvent> gestionnaireSouris =
    evenement -> System.out.println("clické !");

// ajout du gestionnaire
bouton.addEventHandler(MouseEvent.MOUSE_CLICKED, gestionnaireSouris);

// suppression du gestionnaire
bouton.removeEventHandler(MouseEvent.MOUSE_CLICKED, gestionnaireSouris);
```

- On utilise souvent le terme *écouteur* (listener) pour désigner le gestionnaire (handler)

Les événements en JavaFX : méthodes de convenances

Pour les événements les plus courants, l'enregistrement des gestionnaires peut être simplifié :

```
// création d'un événement
EventHandler<MouseEvent> mouseEventHandler =
    evenement -> System.out.println("événement de souris");

// ajout de l'écouteur à la scène avec une méthode de convenance
scene.setOnMouseClicked(mouseEventHandler);
```

- les méthodes `setOnXXX()` (par ex. `setOnMouseClicked()`, `setOnKeyTyped`, ...) sont dites *de convenances*
- ne permettent pas d'attacher plusieurs gestionnaires
- existent uniquement pour les événements les plus courants pour un type de Node donné