UEF4.3.Programmation Orientée Objet

s_sadeg@esi.dz, n_bousbia@esi.dz
Ecole nationale Supérieure d'Informatique (ESI)

Chapitre V

initiation à la Programmation d'interfaces graphique

Interface graphique

- Une interface graphique (GUI pour Graphical User Interface) est formée d'une ou plusieurs fenêtres contenant divers composants graphiques (widgets) tels que :
  - boutons,
  - menus,
  - champs texte,
  - liste déroulante,
  - cases à cocher…etc.

Programmation évènementielle

- L'utilisateur de l'interface graphique interagit avec les objets qui la composent en effectuant des actions (déplacement, clic de souris, frappe de touche de clavier…etc)

- Ces actions déterminent le cheminement du programme, donc l'ordre d'exécution des instructions ne peut être prévu à l'écriture du code. Ce type de programmation est dit **« programmation conduite par les évènements »** ou **« programmation évènementielle »**

Les différents composants d'une application graphique

- Une application graphique est composée de trois parties:
  - **Les composants graphiques**: ils composent l'interface graphique de l'application
  - **Les méthodes « écouteurs » d'évènements** : font appel aux méthode d'application pour répondre aux évènements.
  - **Les méthodes d'application**: reçoivent les données de la GUI et effectuent un traitement puis renvoient les résultats à la GUI pour être affichées

Les principaux concepts de la programmation graphique

La programmation graphique est basée sur les trois concepts suivants:

- Les composants graphiques
- La gestion d'emplacement (ou de positionnement)
- La gestion des évènements

Les principaux concepts de la programmation graphique

Les composants graphiques

Ils sont de deux types:

- Les conteneurs (containers): destinés à contenir d'autres composants graphiques. La fenêtre est un conteneur.
- Les composants simples ou élémentaires qui doivent être contenus dans des conteneurs ex: Boutons, menus, cases à cocher…etc.

Les principaux concepts de la programmation graphique

Les gestionnaires d'emplacement

- Une interface graphique est souvent composée de plusieurs composants.
- La gestion de leur emplacement est déléguée à un gestionnaire qui s'occupe de
  - Calculer leur position
  - Adapter leurs tailles selon la taille de la fenêtre
  - …etc

Les principaux concepts de la programmation graphique

La gestion des évènements

Pour gérer les évènements générés par les actions de l'utilisateur sur l'interface graphique, java utilise des **écouteurs** (*Listeners* en anglais)

- Les composants graphiques (boutons, listes, menus,…) sont écoutés (observés)
- Chaque composant graphique a ses écouteurs *qui écoutent* un type d'événement (par exemple, clic de souris).
- Dès qu'un évènement survient, l'écouteur exécute l'action appropriée

Les GUI avec Java: bref historique

- Au début (1995)les interfaces graphiques étaient créées en utilisant AWT (Abstract Window Toolkit) (java.awt)
  - Composants lourds basés sur la plateforme
- Rapidement (1996), ils ont été remplacés par les composant swing (javax.swing) , plus légers et plus esthétiques.
- JavaFX a ensuite été proposé (2007)et amélioré progressivement jusqu'à son intégration dans la version java 8 (2014) comme étant le nouveau standard pour le développement d'applications graphiques.

Initiation à javaFX

Introduction

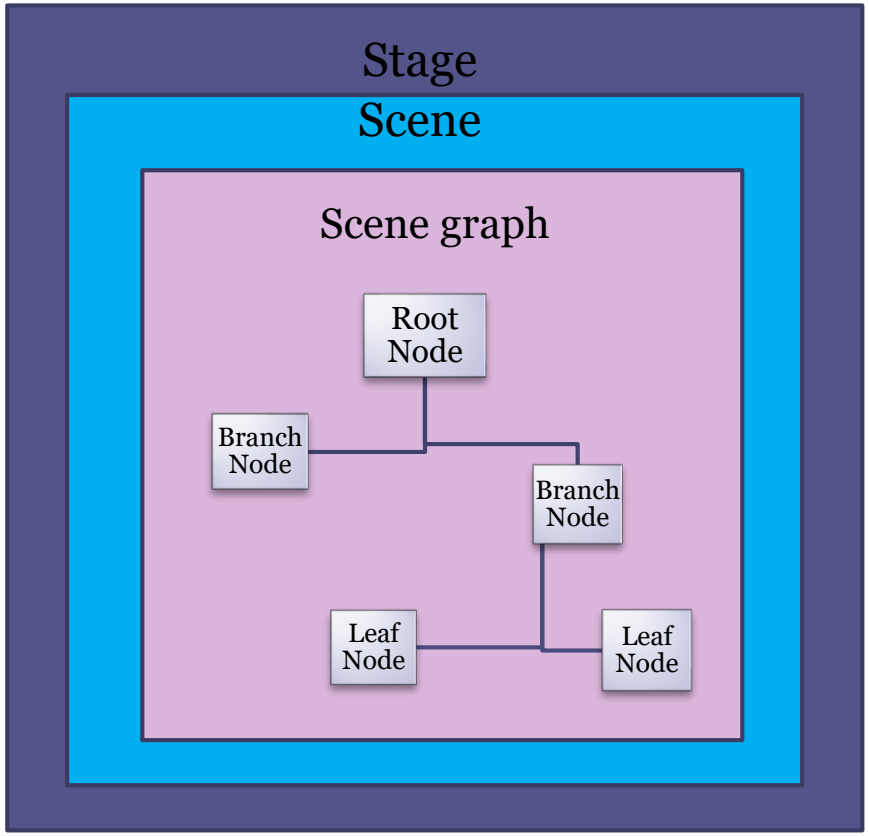
- JavaFX est le nouveau standard Java venu remplacer Swing pour le développement d'applications graphiques riches pouvant utiliser des données variées et être exécutées sur différentes plateformes.

Introduction

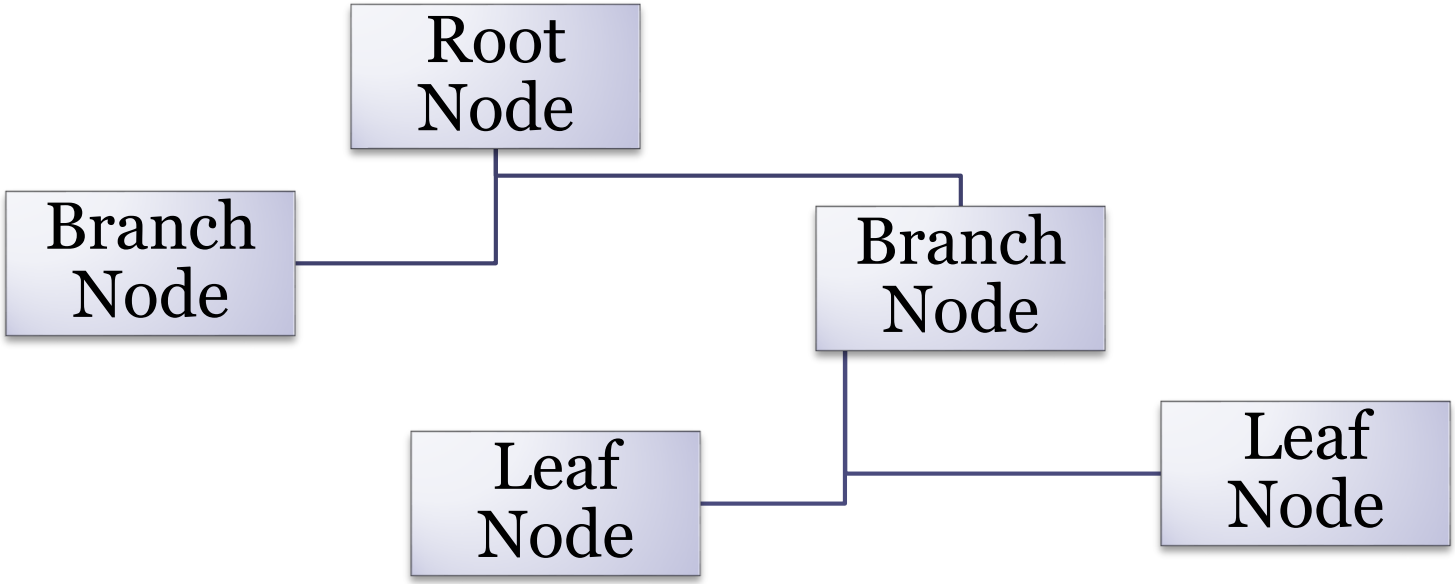
- Avec JavaFX, les interfaces peuvent être créées de deux manières :
  - En écrivant du code Java qui fait appel aux API de la plateforme et qui utilise les composants/conteneurs offerts (classes et interfaces) (approche adoptée dans ce cours)
  - En décrivant l'interface dans un fichier au format FXML qui sera ensuite chargé dynamiquement dans l'application

Structure basique d'un programme JavaFX

- La structure d'une application graphique JavaFX imite celle d'un théâtre . Elle comporte un théatre (*stage*, le niveau le plus élevé), une scène (*Scene* à l'intérieur d'un *Stage*) et un graphe de scène qui représente la hiérarchie des composants (considérés comme des acteurs) dans une scène.



Graphe de scène (scene graph) (1/3)



Graphe de scène (scene graph) (2/3)

- Ces nœuds sont ordonnés de manière hiérarchique.
  - La racine (root node): seul nœud à n'avoir pas de parent. C'est en général un panneau destiné à contenir tous les autres nœuds du graphe
  - Les feuilles du graphe (leaf node): sont des nœuds qui n'ont pas de nœuds fils. Ce sont en général des composant visibles.
  - Les nœuds intermédiaires: ce sont des éléments généralement invisibles qui servent à la mise en place des composants (comme les panneaux)

Graphe de scène (scene graph) (3/3)

- Il comporte tous les éléments graphiques d'une GUI qui sont appelés *noeuds* et héritent de la classe Node.
- Un nœud peut être:
  - Une forme géométrique 2D ou 3D comme un cercle, un rectangle …etc
  - Un composant comme un bouton, une zone de texte, une case à cocher..
  - Un conteneur
  - Un élément média comme une vidéo, une image..etc

Première fenêtre

Structure basique d'un programme JavaFX

- Tout programme javaFX est défini dans une classe dérivant de **javafx.application.Application** et redéfinissant sa méthode abstraite **start()**

- Dans la méthode main, il suffit de lancer l'application en invoquant la méthode **launch ()** qui appelle à son tour la méthode **start()**

Structure basique d'un programme JavaFX

```
import javafx.application.*;
import javafx.stage.*;
import javafx.scene.*;

public class PremiereFenetre extends Application{

public void start(Stage primaryStage)  {
      /*
      Code de l'application JavaFX
      (Stage, scene, scene graph)
      */
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

Structure basique d'un programme JavaFX

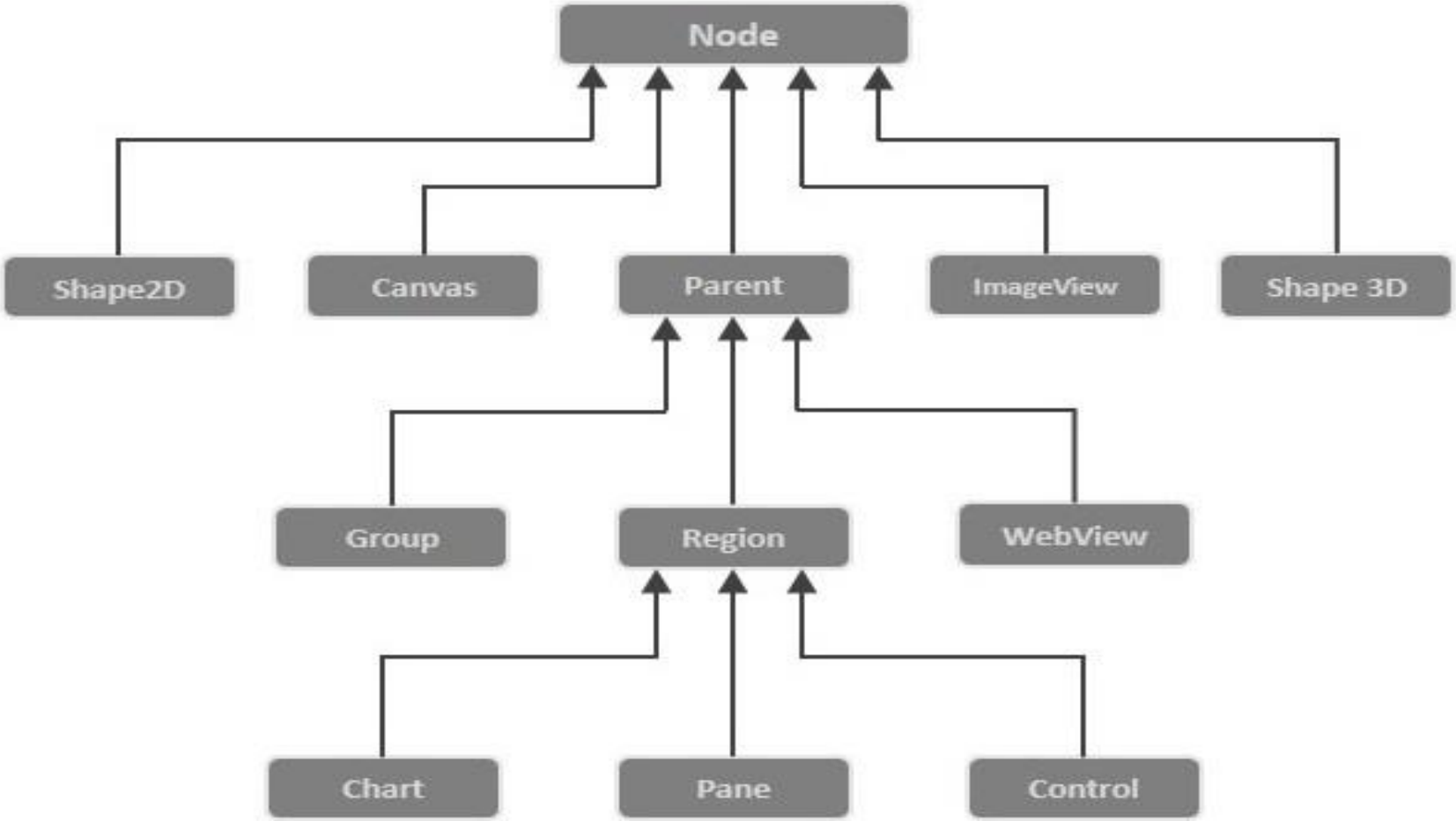
Au sein de la méthode start, il faut :

1. Préparer le graphe de scène
2. Préparer la scène en définissant ses dimensions et y ajouter le graphe de scène (le noeud root)
3. Préparer le théâtre (Stage) et y ajouter la scène

Première fenêtre en JavaFX

1. Préparer le graphe de scène

- Il faut préparer le graphe de scène avec les noeuds nécessaires. Pour cela il faut d'abord créer le noeud racine (root node) qui peut être:
  - Un objet Group
  - Un objet Region (un chart, un panneau, un control…)
  - Un objet WebView



Hiérarchie de la classe Node

```
public class ExempleJavaFX extends Application {
public void start(Stage primaryStage) {
     //créer un objet Group qui sera la racine
     Group root = new Group();
 //Créer une scène en passant l'objet Group, la hauteur et la largeur
     Scene scene = new Scene(root ,600, 300);
 //colorer la scène (facultatif)
scene.setFill(Color.BLUE);
 //Donner un nom au Stage (facultatif)
     primaryStage.setTitle("Sample Application");
 //ajouter la scène au Stage
     primaryStage.setScene(scene);
 //Rendre le contenu du Stage visible (montrer le Stage)
     primaryStage.show();
  }
  public static void main(String args[]){
     launch(args);
  }
}
```

Gestion de la mise en forme (Layout Managers)

Introduction

- L'utilisateur d'une interface graphique peut changer la taille d'une fenêtre, mais cela ne doit pas modifier son aspect. Les composants graphiques doivent alors être repositionnés et /ou redimensionnés.

- Cette tâche est déléguée au gestionnaire de mise en forme (layout manager) utilisé par la fenêtre (ou autre conteneur)

- Il existe plusieurs types de *layout managers* avec des algorithmes (politiques) de placement différents

Tailles des composants

- Tous les composants graphiques ont plusieurs types de tailles qui sont utilisées pour leur affichage
  - taille maximum
  - taille préférée
  - taille minimum
- Ces paramètres peuvent être récupérés ou spécifiés en utilisant les Accesseurs et modificateurs associés

`{get|set}{Max|Pref|Min}{Size|Height|Width}`

Quelques Composants graphiques

Layout manager

- A l'ajout d'un composant, l'utilisateur ne précise pas l'emplacement de celui-ci, il se contente juste de donner des indications propres au *layout manager* utilisé par le conteneur
- Celui-ci (le *layout manager)* met en forme les composants « *au* mieux » (de la meilleure manière possible) suivant :
  - l'algorithme de placement qui lui est propre
  - les indications de positionnement des composants
  - la taille du conteneur
  - les tailles préférées des composants

Layout managers

- BorderPane: place les éléments dans 5 régions.
- HBox : place les éléments horizontalement sur une ligne.
- VBox : place les éléments verticalement sur une colonne
- StackPane: place les éléments suivant une pile ou chaque élément s'ajoute audessus du précédent
- GridPane : permet de placer les éléments sur une grille de manière flexible
- FlowPane: place les éléments horizontalement ou verticalement selon la limite fixée de la largeur ou de la hauteur
- TilePane: place les éléments de la même manière que FlowPane en leur attribuant des emplacements de même taille
- AnchorPane: permet d'ancrer les éléments en haut, en bas, à droite, à gauche ou au centre du conteneur

Layout managers

- Exemple avec 5 boutons

Layout managers

- Pour obtenir la mise en forme désirée, il est souvent utile de combiner plusieurs Layout managers au sein d'une même application JavaFX

Gestion des évènements

Introduction

- Dans une application graphique, l'utilisateur interagit avec la GUI en utilisant, par exemple, le clavier et la souris.
- Chaque action de l'utilisateur engendre un évènement qui impacte le déroulement du programme.
- Pour cela, les évènements doivent être liés à des traitements dans le programme.

Les évènements en JavaFX

- Un évènement en JavaFX est un objet de la classe javafx.event.Event ou une de ses classes dérivées.

Exemples:

- **MouseEvent** : évènement créé lorsque l'utilisateur manipule la souris
- **KeyEvent** : évènement indiquant qu'une touche du clavier a été manipulée par l'utilisateur
- **Drag Event:** évènement déclenché lorsque la souris est glissée
- **Window Event:** évènements liés à la fenêtre et se produisant lorsqu'une fenêtre est cachée ou rendue visible par exemple

Les évènements en JavaFX

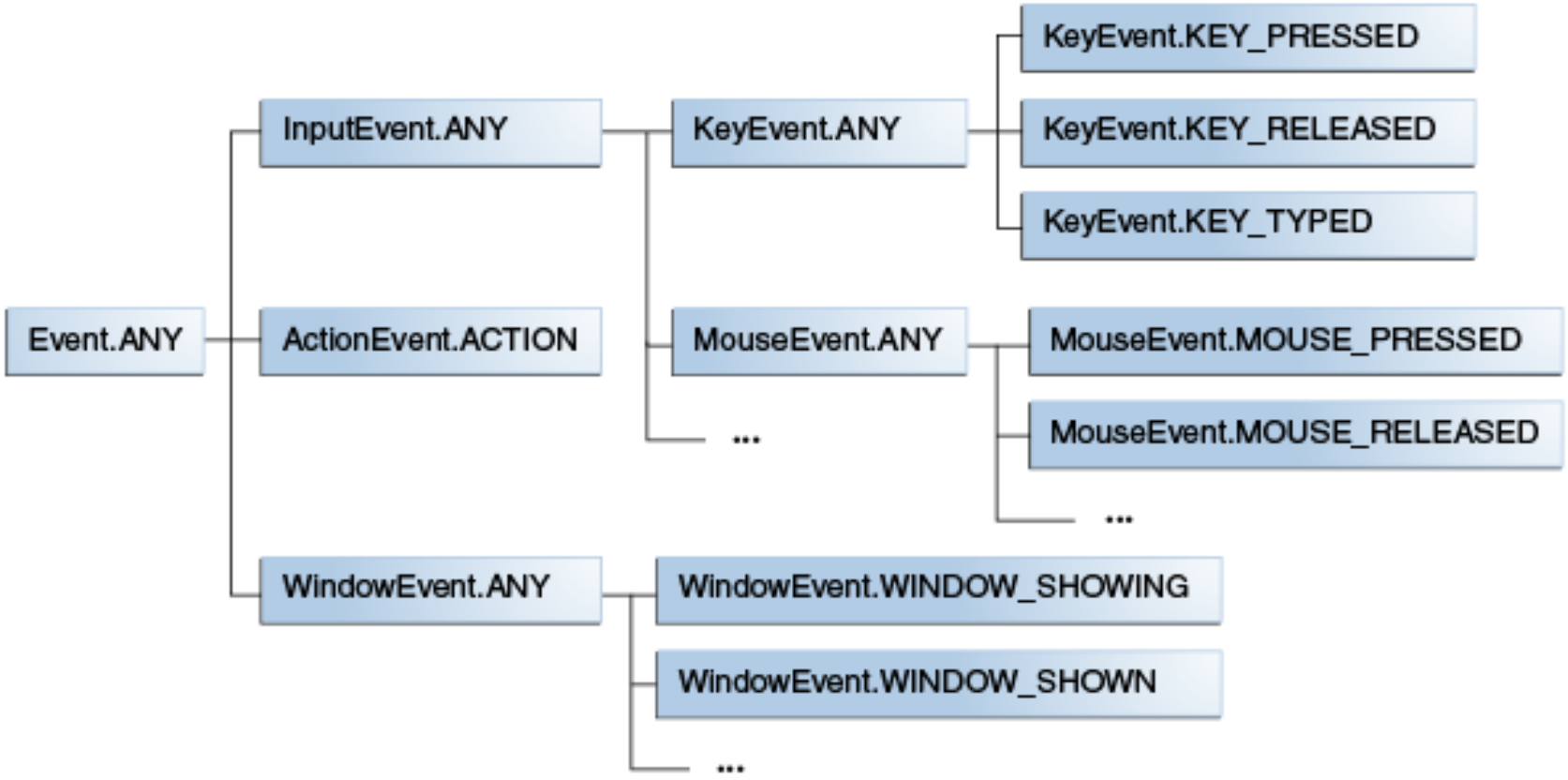
- En JavaFX chaque évènement possède:
  - **Une cible (target):** le noeud sur lequel l'évènement s'est produit, cela peut être une fenêtre, une scène ou un noeud.
  - **Une source :** La source de l'évènement est celle qui génère l'évènement. Par exemple, la souris.
  - **Type :** C'est le type de l'évènement. Dans le cas des évènements liés à la souris le type peut être: mouse pressed, mouse released …

Types d'évènements

- Un type d'évènement est une instance de la classe EventType.
- Les types d'évènements classifient les évènements d'une classe d'évènements. Par exemple, la classe KeyEvent contient les types suivants:
  - KEY_PRESSED
  - KEY_RELEASED
  - KEY_TYPED

- Les types d'évènements sont hiérarchisés. Chaque évenment a un nom (getName()) et un super type (getSuperType()). par exemple, L'évènement relatif à une touche appuyée est KEY_PRESSED, et son super type est KeyEvent.ANY.

Types d'évènements



Une partie de la hiérarchie des types d'évènement

Cibles de l'évènement

- La cible d'un évènement est une instance d'un évènements de n'importe quelle classe implémentant l'interface EventTarget
- Cette intreface dispose de la méthode buildEventDispatchChain qui crée le chemin de traitement des évènements (event dispatch chain) dans le graphe de scène que l'évènement doit parcourir pour atteindre la cible
- Les classes Window, Scene, et Node implémentent l'interface EventTarget et leurs sous classes héritent de cette implémentation.

Gestion des évènements en JavaFX

Pour gérer les évènements, JavaFX fournit des “handlers” et des “filters”. Ce sont des parties de code qui sont exécutées lorsqu’un évènement survient.

Le traitement d’un évènement passe par 3 étapes:

1. Sélection de la cible (Target selection)
2. Construction de la chaine de traitement(Route construction)
3. Parcours de la chaine de traitement
   - 3.1. Traitement des filtres de l’évènement (Event capturing)
   - 3.2. Traitement des gestionnaires de l’évènement (Event bubbling)

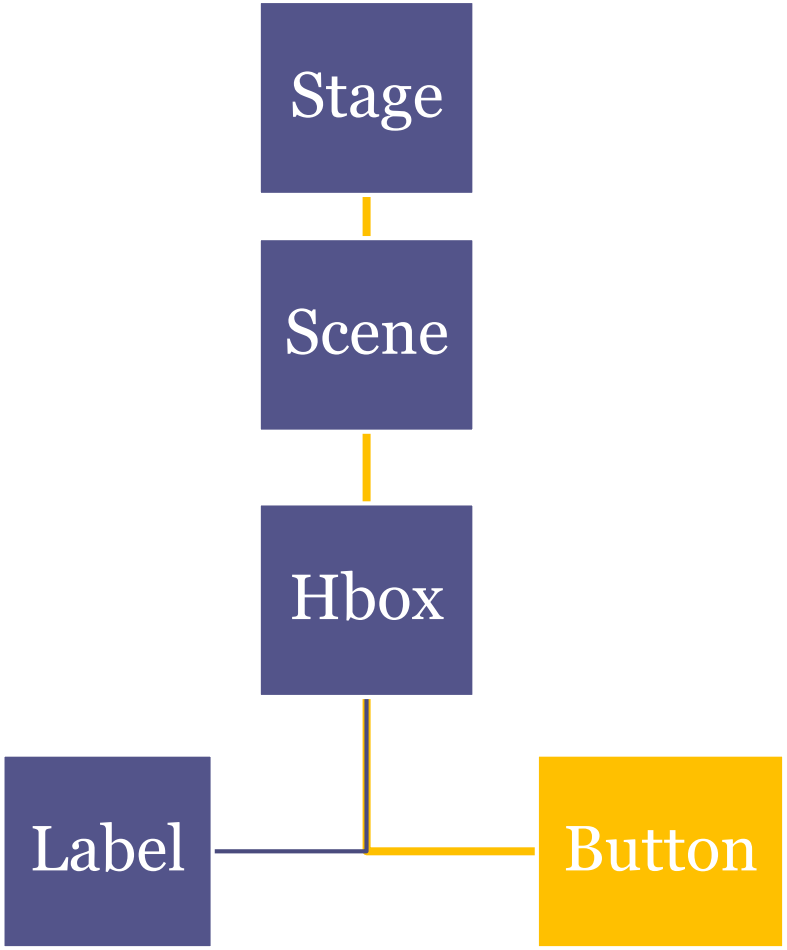
Etapes de traitement d'un évènement
1. Sélection de la cible (Target selection)

- La cible d'un évènement est le noeud d'arrivée (destination) d'un évènement
- La cible d'un évènement est selectionnée selon le type de l'évènement
  - évènements de la souris: la cible est le noeud(composant) sur lequel se trouve le curseur
  - Évènements du clavier: la cible est le noeud qui a le focus
  - Gestes de glissement: la cible est le composant au centre de la position des doigts ou de la souris
  - Évènements tactiles: la cible est le composant qui a été pressé en premier

Remarque: Si plus d'un composant se trouve au centre c'est celui qui se trouve le plus haut qui est considéré comme étant la cible.

Etapes de traitement d'un évènement
2.Construction de la chaîne de traitement

- Le chemin part de la racine *(Stage) et va jusqu'au composant cible en* parcourant tous les noeuds intermédiaires.



Etapes de traitement d'un évènement
3.Parcours de la chaine de traitement

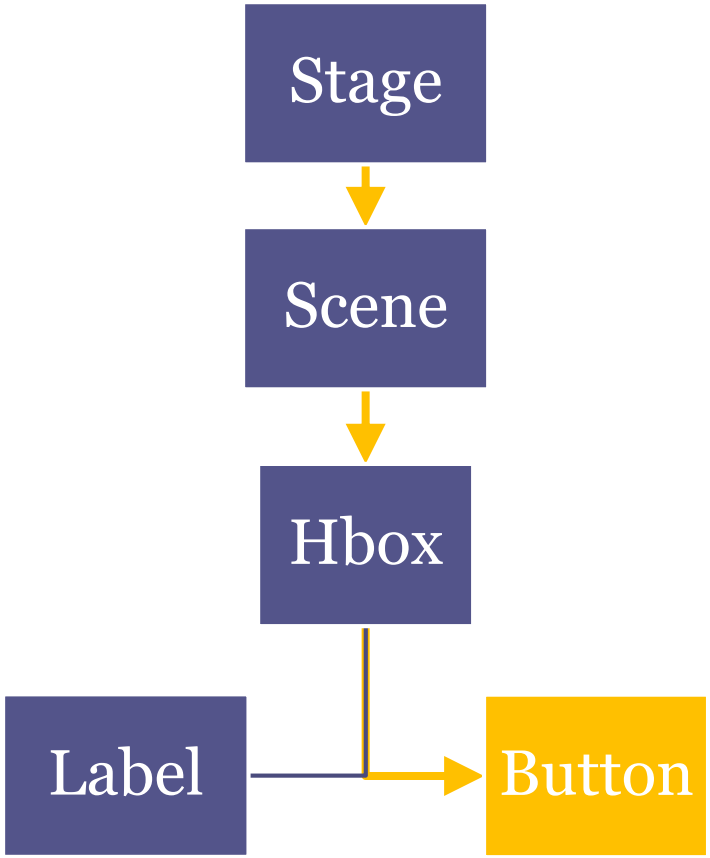
- Un évènement passe par chaque nœud du chemin de traitement des évènements deux fois: une fois durant la phase de traitement des filtres (capturing phase) et une autre fois durant la phase de traitement des gestionnaires (bubbling phase).
- Les filtres (filters) enregistrés pour un nœud sont exécutés durant la phase descendante(capturing phase)
- Les gestionnaires (handlers) enregistrés sont exécutés durant la phase montante(bubbling phase)

Etapes de traitement d'un évènement

3.Parcours de la chaine de traitement

3.1 Traitement des filtres de l'évènement (Event capturing)

- Exécute le code des filtres(event filters) en suivant le chemin descendant, de la racine *(Stage)* *jusqu'au composant cible*

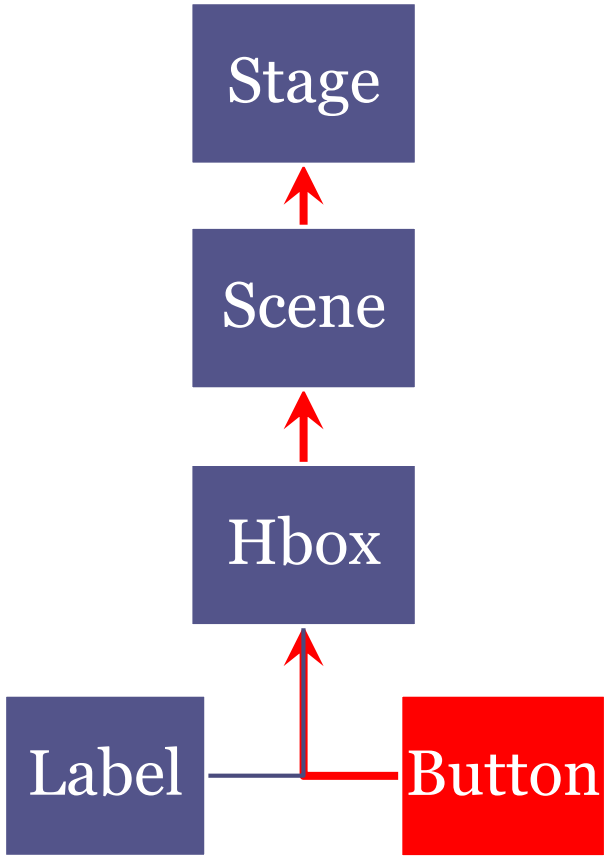


Etapes de traitement d'un évènement

3.Parcours de la chaine de traitement

3.2 Traitement des gestionnaires de l'évènement (Event bubbling)

- Exécute le code des gestionnaires (event handlers) en suivant le chemin montant, de la cible jusqu'à la racine *(Stage)*.



Gestion de l'évènement (1/3)

- Pour gérer un événement (exécuter les instructions qui lui sont associées), il faut créer un **écouteur d'événement *(Event Listener)*** et l'**enregistrer** sur les nœuds du graphe de scène où l'on souhaite intercepter l'événement et effectuer un traitement.

- Un écouteur d'événement peut être enregistré comme **filtre** ou comme **gestionnaire d'événement.**

- Si un noeud du graphe de scène possède plusieurs écouteurs d'événements enregistrés, l'ordre de leur exécution dépendra de la hiérarchie des types: un écouteur pour un **type spécifique est toujours exécuté avant un écouteur pour un type plus générique**

- **Exemple:** un filtre enregistré pour MouseEvent.MOUSE_PRESSED sera exécuté avant un filtre pour MouseEvent.ANY qui sera exécuté avant un filtre pour InputEvent.ANY

Gestion de l'évènement (1/3)

- Les filtres et les gestionnaires d'événements, sont des objets implémentant l'interface **EventHandler<T extends Event>**contenant une seule méthode **handle(T event)** qui se charge de traiter l'événement.

```
public interface EventHandler<T extends Event> extends
EventListener
        void handle(T event);
}
```

Gestion de l'évènement (1/3)

- Pour **enregistrer un écouteur d'événement sur un noeud du graphe** de scène, on peut :

1) Enregistrer un filtre d'évènement en utilisant la méthode **addEventFilter()** que possèdent tous les noeuds

2) Enregistrer un gestionnaire en utilisant la méthode **addEventHandler()** que possèdent tous les noeuds

3) Utiliser une des **méthodes utilitaires *(convenience methods)*** *dont* disposent certains composants et qui permettent d'enregistrer un gestionnaire d'événement (handler) en tant que propriété du composant. Ces méthdes sont disponibles, notamment, dans la classe Node (et ses dérivées). Elles sont nommées **setOn*EventType(EventHandler)*, *par exemple :***
   - **setOnAction(*Handler*)**
   - **setOnKeyPressed(*Handler*)**

Consommation d'un évènement

- Un évènement peut être consommé par un filtre ou un handler à n'importe quel noeud de la chaine de traitement (event dispatch chain) en invoquant la méthode consume(). Cette méthode indique que le traitement de l'évènement est terminé.
- Plusieurs filtres (resp. handlers) peuvent être enregistrés pour un noeud.
- Si un noeud consomme un évènement dans un de ses filtres ou handlers en invoquant la méthode **consume()**, les autres filtres et handlers sur le même noeud sont exécutés et le traitement de l'évènement s'arrête sans parcourir les noeuds restants du chemin.
- **Remarque**: Par défaut, les handlers pour les composants javaFX consomment la plupart des évènements inputEvent

Exemple complet sur machine