Université Constantine 2
جامعة قسنطينة 2

IHMI

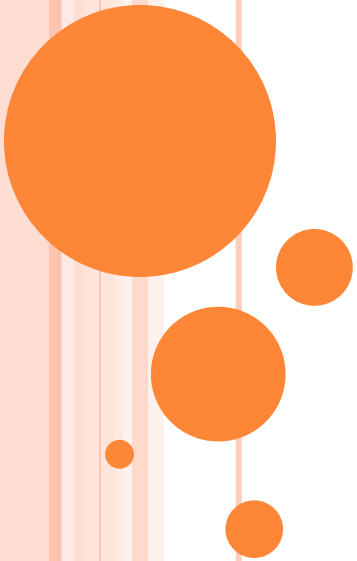
– Cours 7 –

Chapitre 6: Architecture MVC et conception d'interfaces

Dr. Mostefai Sihem
Faculté des nouvelles technologies
sihem.mostefais@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	IFA	Licence 3	Technologies de l'information (TI)



Chapitre 6: Architecture MVC et conception d'interfaces

Plan

- Structure d'une application
- Principe de l'architecture MVC
- Interactions dans le MVC
- Composants du MVC
- Le rôle du contrôleur et la gestion des évènements
- Exemple applicatif

Structure d'une application

- En conception d'interfaces, on peut structurer le code d'une application de plusieurs façons.
- On peut avoir un fichier unique contenant le code de l'application ➔ source de plusieurs problèmes
- **MVC = Model - View – Controller**, est un modèle d'architecture pour bien structurer l'application.
  - le code de l'application est divisé en entités distinctes qui communiquent entre- elles
- **MVC** est un modèle de conception, **design pattern**, qui a été introduit avec le langage Smalltalk-80
  - Pour simplifier le développement et la maintenance des applications, en répartissant et en découplant les activités dans différents sous- systèmes (plus ou moins) indépendants.

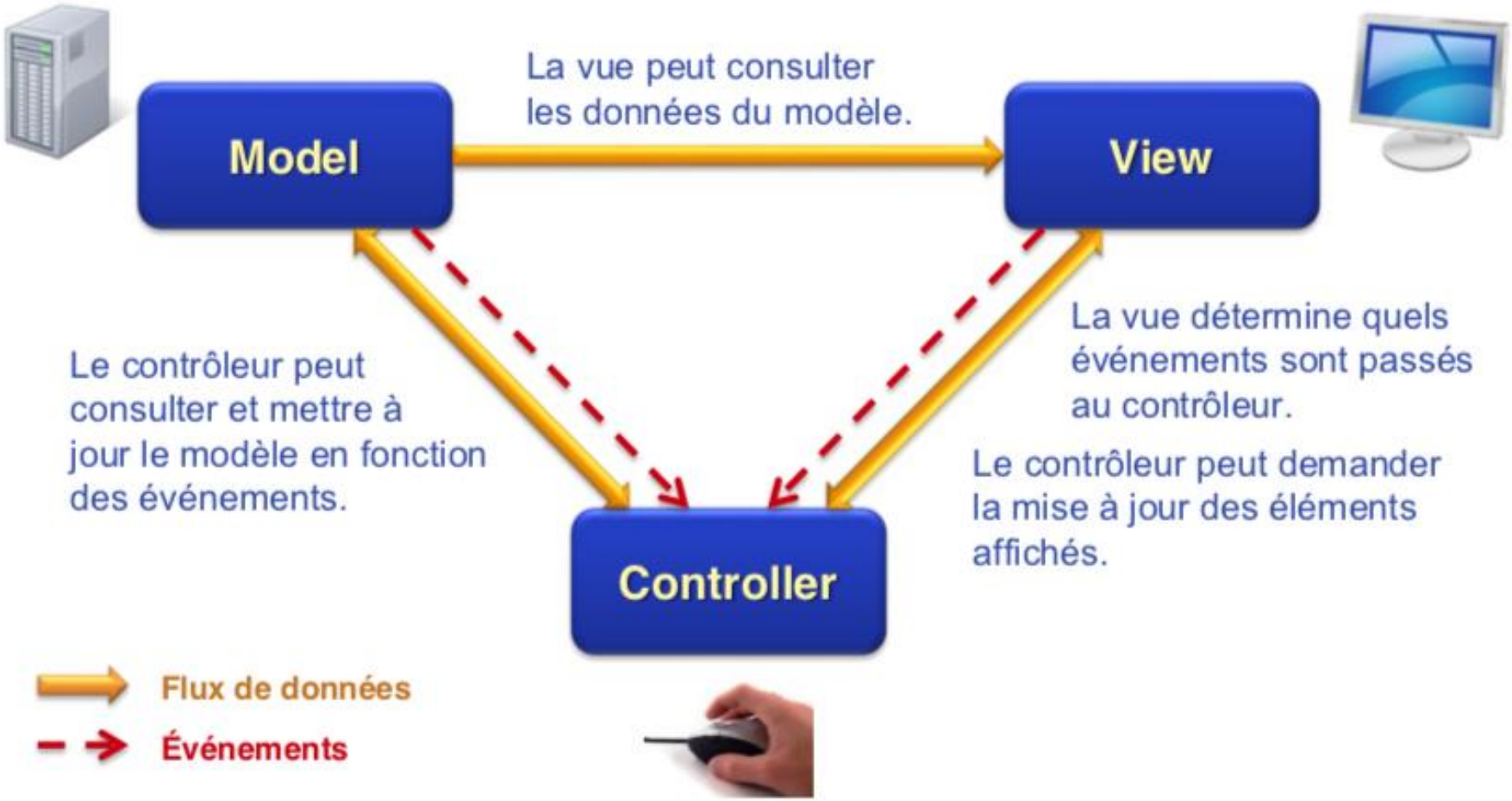
Principe de l'architecture MVC

Le code de l'application est divisé en trois parties distinctes :

- Le **modèle** (**Model**) qui se charge de la gestion des données et contient la logique applicative (accès, transformations, calculs, etc.).
- Les **vues** (**Views**) qui représentent les interfaces utilisateur (composants, fenêtres, boîtes de dialogue) de l'application.
- Les **contrôleurs** (**Controllers**) qui sont chargés de réagir aux actions de l'utilisateur (clavier, souris, gestes) et à d'autres événements internes et externes .

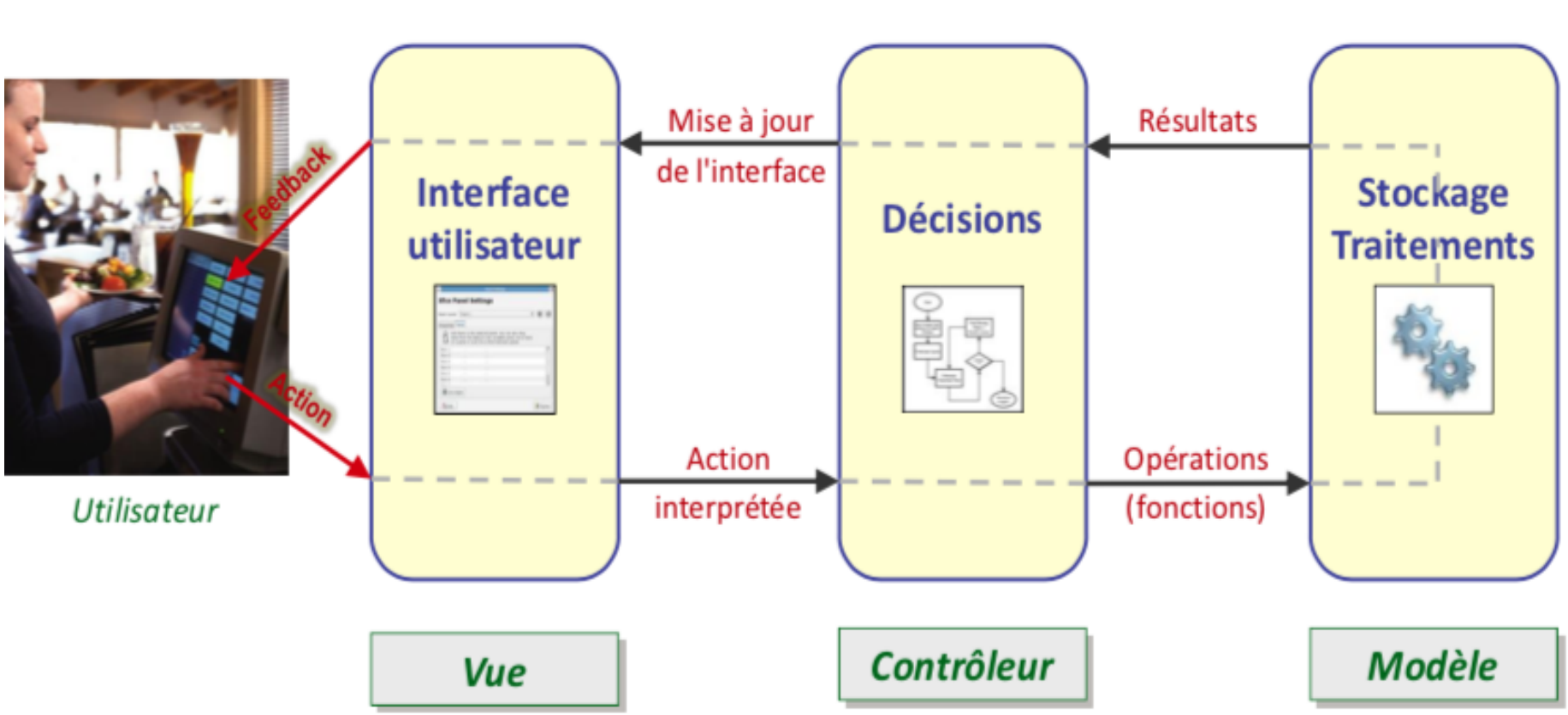
Une application peut comporter du code qui n'est pas directement affecté à l'une de ces trois parties (librairies générales, classes utilitaires, etc.).

Interactions dans le MVC



Intéractions et rôle de l'interface

- L'utilisateur intéragit par actions et feedbacks



Rôle des composants MVC
Le Modèle

- **Le modèle (Model)** est responsable de la gestion des données de l'application.
- Le modèle peut contenir lui- même les données ou etre un intermédiaire (proxy) qui gère l'accès aux données (dans une BD, un serveur, le cloud, ...)
- En Java, il est souvent défini par une ou plusieurs interfaces pour manipuler les donnees (les objets métier) independemment de leur stockage (notion de DAO Data Access Object).
- Les informations gérées par le modèle doivent être **indépendantes** de la façon de les afficher. Le modèle doit pouvoir exister indépendamment de la représentation visuelle des données.

Rôle des composants MVC
La Vue

- **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.
- La vue récupère certaines actions de l'utilisateur et les transmet au contrôleur pour qu'il les traite (souris, clavier, gestes, ...).
- La mise à jour de la vue peut être déclenchée par un contrôleur ou par un événement signalant un changement intervenu dans les données du modèle par exemple (mode asynchrone).

Rôle des composants MVC
Le Contôleur

- **Le contrôleur (Controller)** est chargé de réagir aux différentes actions de l'utilisateur et d'autre événements pouvant survenir.
- Il définit le comportement de l'application et sa logique (comment elle réagit aux évènements de la vue).
- Dans les applications simples, le contrôleur gère la synchronisation entre la vue et le modèle (rôle de chef d'orchestre).
- il existe un couplage assez fort entre la vue et le contrôleur.
- Le contrôleur communique avec le modèle et avec la vue.

Variantes du MVC

- Il existe de nombreuses variantes de l'architecture MVC dont les plus connues sont :
  - MVP : Model - View – Presenter
  - MVVM : Model - View - View-Model
  - MV- Whatever
- Dans ces variantes le modèle et la vue jouent leurs rôles comme dans le MVC standard. C'est le rôle du contrôleur et sa manière de communiquer avec les autres parties qui caracterisent ces variantes de l'architecture MVC.

Mise en oeuvre MVC en JavaFX

Une application JavaFX qui respecte l'architecture MVC comporte :

- Le **modèle** sera souvent représenté par une ou plusieurs classes qui gerent les données de l'application.
- Les **vues** seront soit codées en Java ou **déclarées en FXML**. On peut avoir aussi des feuilles de styles.
- Les **controleurs** pourront être :
  - Des classes qui traitent chacune un événement particulier ou qui traitent plusieurs evénements relies
  - 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 separée ou être integrée à la classe de la fenêtre principale.
- D'autres **classes utilitaires** peuvent venir compléter l'application.

Concrétisation du MVC en JavaFX

- Dans une application JavaFX:
  - Vue = Les classes qui dessinent l'interface
  - Modèle = Les classes de la logique applicative
  - Contrôleur = Les classes qui gèrent les évènements générés par les composants de l'interface

  Le contrôleur joue le **rôle le plus important** en traitant les évènements générés par la Vue

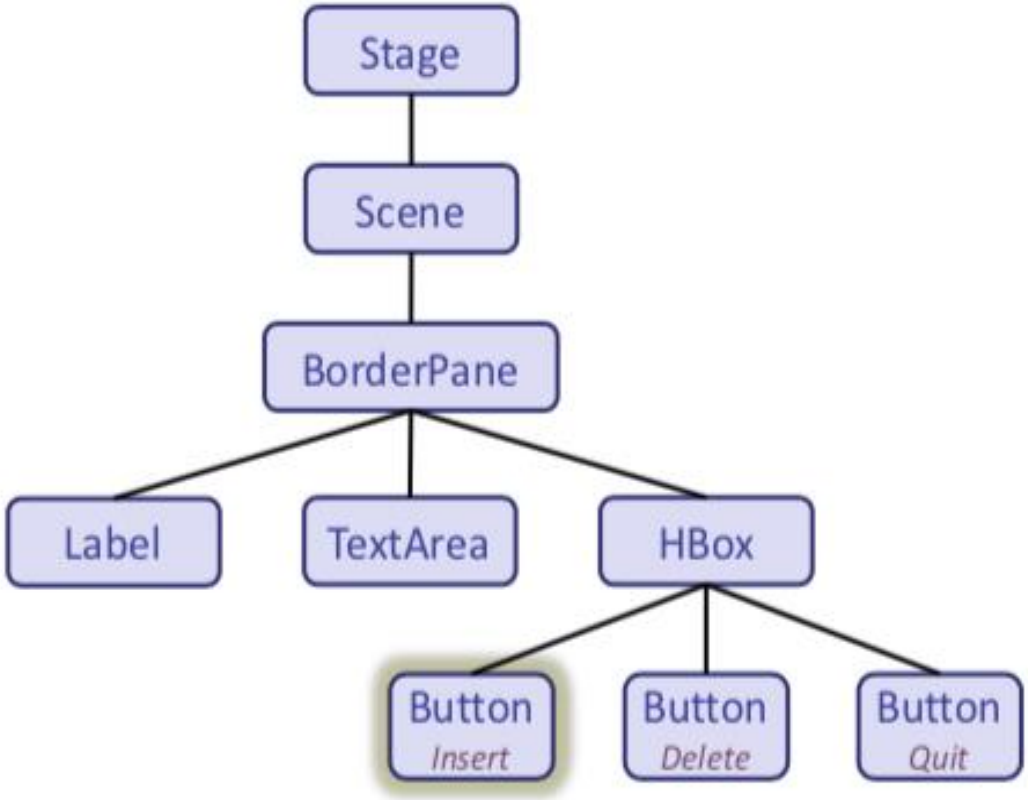
Gestion des évènements (1)

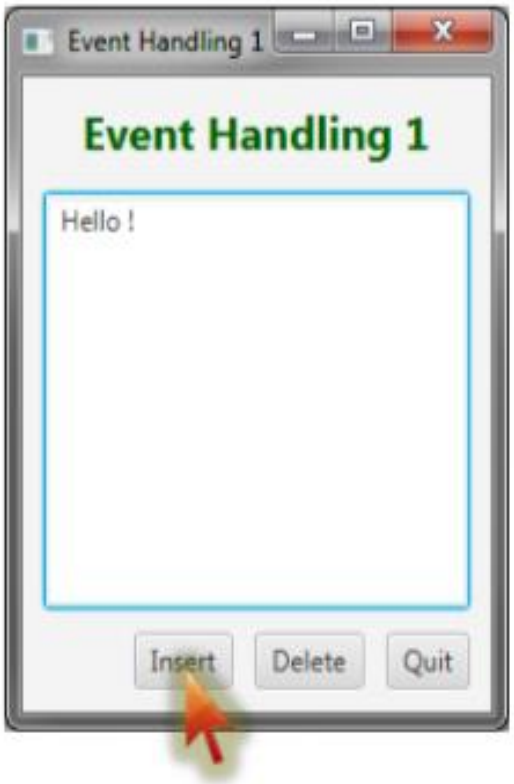
Le traitement des évènements implique les étapes suivantes :

- **Sélection de la cible** (Target) de l'évènement
  - Événement clavier ➔ le composant qui possède le focus
  - Événement souris ➔ le composant sur lequel se trouve le curseur
  - etc
- **Détermination de la chaîne de traitement** des événements (Event Dispatch Chain : chemin des événements dans le **graphe de scène**)
- **Traitement des filtres d'événement** (Event Filter)
  - Exécute le code des filtres en suivant le chemin descendant, de la racine (Stage) jusqu'au composant cible
- **Traitement des gestionnaires d'événement** (Event Handler)
  - Exécute le code des gestionnaires d'événement en suivant le chemin montant, du composant cible à la racine (Stage)

Gestion des évènements (2)

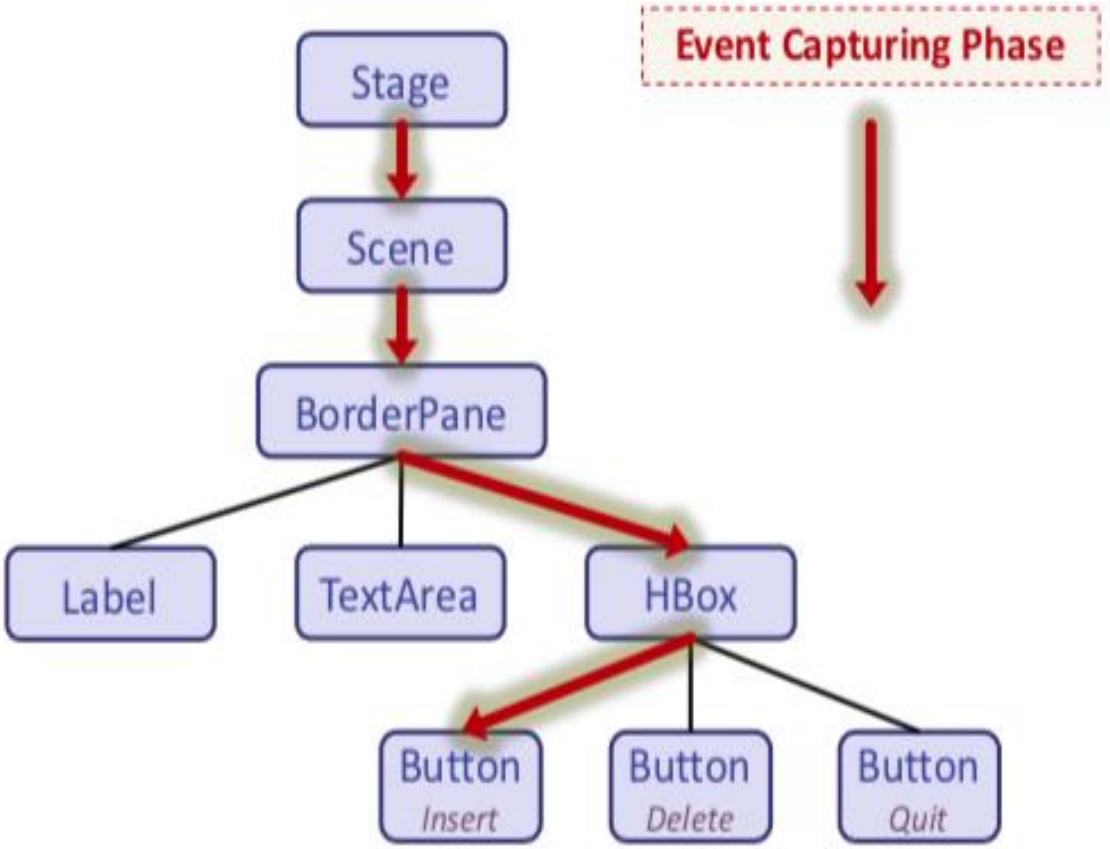
- Un exemple d'application avec son graphe de scène.
- Si l'utilisateur clique sur le bouton *Insert*, un événement de type *Action* va être déclenché et va se propager sur le chemin correspondant à la chaîne de traitement (Event Dispatch Chain).





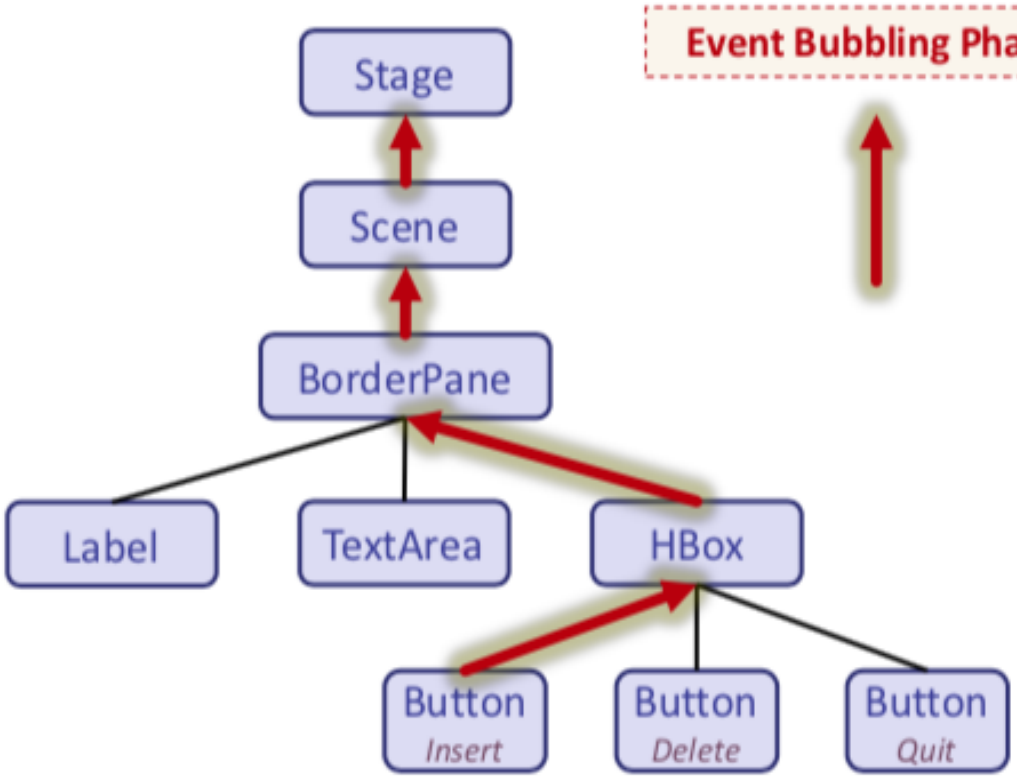
GESTION DES ÉVÈNEMENTS (3)

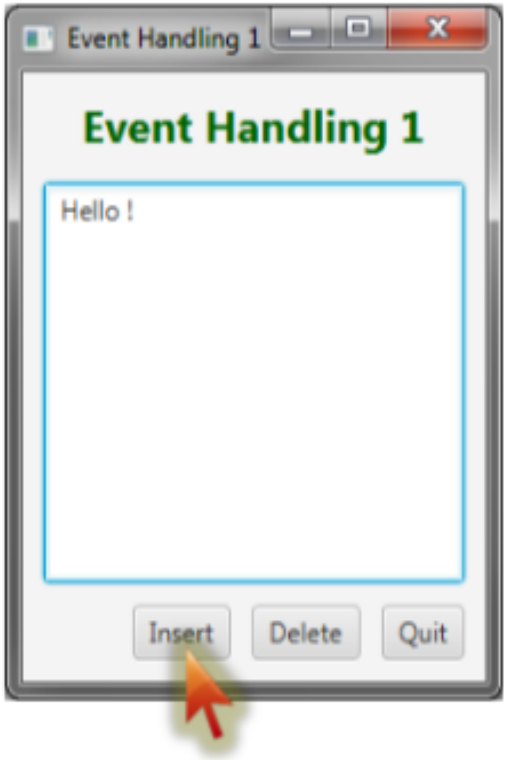
- L'événement se propage d'abord vers le bas, depuis le nœud racine (Stage) jusqu'à la cible (Target) - c'est-à-dire le bouton cliqué - et les filtres (Event Filter) éventuellement enregistrés sont exécutés (dans l'ordre de passage).



GESTION DES ÉVÈNEMENTS (4)

- L'événement remonte ensuite depuis la cible jusqu'à la racine et les gestionnaires d'événements (Event Handler) éventuellement enregistrés sont exécutés (dans l'ordre de passage).





Gestion des évènements (5)

- Pour traiter un événement, il faut créer un **récepteur d'événement (Event Listener)**, ou écouteur, et l'enregistrer sur les nœuds du graphe de scène où l'on souhaite intercepter l'événement et effectuer un traitement.
- Un récepteur d'événement peut être enregistré comme **filtre** ou comme **gestionnaire d'événement**.
  - **Les filtres (filters)** sont exécutés dans la phase descendante de la chaîne de traitement des événements (avant les gestionnaires)
  - **Les gestionnaires (handlers)** sont exécutés dans la phase montante de la chaîne de traitement des événements (après les filtres)
- Les filtres et les gestionnaires d'événements, sont des objets qui doivent implémenter l'interface fonctionnelle (générique)

EventHandler <T extends Event>

contenant l'unique méthode ***handle(T event)*** qui se charge de traiter l'événement.

Gestion des évènements (6)

Pour enregistrer un récepteur d'événement sur un nœud du graphe de scène, on a 2 possibilités:

- Soit par la méthode ***addEventFilter()*** sur ce nœud et qui permet d'enregistrer un filtre
- Soit par la méthode ***addEventHandler()*** sur ce nœud et qui permet d'enregistrer un gestionnaire d'événement
- Soit par des méthodes utilitaires sur certains composants et qui permettent d'enregistrer un gestionnaire d'evénement (en tant que propriété du composant).

La plupart des composants disposent de méthodes nommées selon le schéma:

- ***setOnEventType(EventHandler),*** par exemple :
  setOnAction(Handler)
  setOnKeyTyped(Handler)

Gestion des évènements (7)

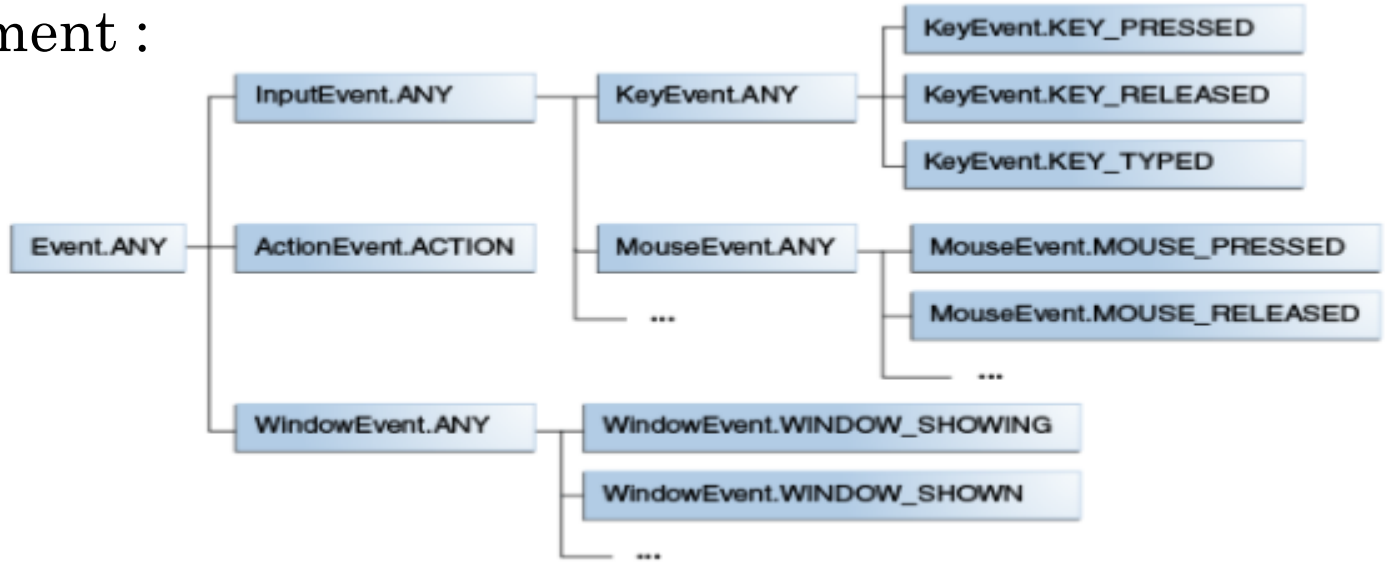
- Les événements se propagent sur la chaîne de traitement du graphe de scène de la racine jusqu'au noeud cible puis l'inverse.
- Sur chaque nœud du graphe de scène on peut enregistrer
  - un ou plusieurs filtres
  - un ou plusieurs gestionnaires d'événements

qui se chargeront de traiter différents types d'événements avant de les propager au nœud suivant en parcourant la chaîne de traitement.

- Chaque récepteur d'événement (filtre ou gestionnaire) peut interrompre la chaîne de traitement en consommant l'événement, en invoquant la méthode ***consume().***
- Si un récepteur d'événement appelle la méthode ***consume()***, la propagation de l'événement s'interrompt et les autres récepteurs (qui suivent dans la chaîne de traitement) ne seront plus activés.

Gestion des évènements (8)

Si un nœud du graphe de scène possède plusieurs récepteurs d'événements enregistrés, l'ordre d'activation de ces récepteurs sera basé sur la hiérarchie des types d'événement :



- Un récepteur pour un type spécifique sera toujours exécuté avant un récepteur pour un type plus générique
- Par 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

Exemple d'application

- On considère l'interface suivante:

- Nous allons illustrer le modèle MVC pour une application simple utilisant cette interface
- Dans ce cas, l'interface est très simple, on n'a pas de modèle.

Exemple d'application
Code de la classe **View**

voir document d'exemple

Exemple d'application
Code de la classe **Controller**

Plusieurs possibilités (Pour traiter les événements des boutons):

1. On peut créer une classe contrôleur qui implémente *EventHandler* et effectue les opérations souhaitées dans la méthode *handle()*.
2. On peut créer le contrôleur sous forme d'une classe locale anonyme.
3. Utiliser la méthode *setOnAction()* et passer en paramètre une expression lambda implémentant la méthode *handle()* de l'interface *EventHandler*.

voir document d'exemple et exercice applicatif en TP.