Interfaces graphiques - JavaFX

responsable : Wiesław Zielonka
zielonka@liafa.univ-paris-diderot.fr
http://liafa.univ-paris-diderot.fr/~zielonka

February 24, 2016

propriétés $\longrightarrow$ listeners

évènements $\longrightarrow$ handlers

Propriétés (beans de javafx)

Une propriété d'un objet (dans le sens de javafx) est une variable privée d'instance de l'objet qui satisfait les conditions suivantes :

- le type de cette variable est un des types défini dans le package `javafx.beans.property` : `BooleanProperty`, `DoubleProperty`, `IntegerProperty`, `ObjectProperty`, `StringProperty` etc.

Évidemment `BooleanProperty` permet de stocker une valeur booléenne, `DoubleProperty` une valeur double etc.

Définir une propriété

- pour chaque propriété la classe définit trois méthodes. Supposons par exemple que la classe possède une variable d'instance

```
private DoubleProperty hauteurMaison =
        new SimpleDoubleProperty();
```

Dans ce cas la classe doit définir les méthodes :

```
public final double
  getHauteurMaison(){ return hauteurMaison.get(); }

public final void
  setHauteurMaison(double v){ hauteurMaison.set(v); }

public DoubleProperty
  hauteurMaisonProperty(){ return hauteurMaison ;}
```

Notez la correspondance entre le nom de la propriété et les noms des méthodes.

Autre exemple

```
public class Model {
    // Les donnees sont constituees d'un seul entier.
    private IntegerProperty entier;

    Model(int val) { entier =
       new SimpleIntegerProperty(val); }

    Model() { this(0); }

    public final int getEntier(){
        return entier.get();
    }
    public final IntegerProperty entierProperty(){
        return entier;
    }

    public final void setEntier(int val){
        entier.set(val);
    }
}
```

Définir ChangeListeners

Pour chaque propriété nous pouvons définir un listener `ChangeListener` qui sera activé quand la valeur de la propriété change. Par exemple un listener pour la propriété du modèle de la page précédente :

```
ChangeListener<Number> modelListener =
    new ChangeListener<Number>() {

  @Override
  public void changed(
        ObservableValue<? extends Number> observable,
        Number old_val,
        Number new_val) {

    //mettre a jours les views quand la propriete
    //du modele change
    for(UpdatableView view : views){
        view.update(model.getEntier());
        //view.update(new_val.intValue());
    }
  }
};
```

Enregistrer ChangeListener pour la propriété du modèle

```
model.entierProperty().addListener(modelListener);
```

Notez que `ChangeListener` était paramétré par un type qui dépend de type de la propriété.

Propriété de contrôles

Les contrôles sont des beans de javafx, par exemple :

`CheckBox`	`BooleanProperty selected`
`ChoiceBox`	`ObjectProperty<T> value`
`Slider`	`DoubleProperty value`
etc	

Ces trois contrôles possèdent aussi d'autres propriétés et tous les contrôles possèdent de propriétés auxquelles on peut attacher `ChangeListeners`.

ChoiceBoxView avec un ChangeListener

```
public class ChoiceBoxView extends ChoiceBox<Integer> {
    private Model model;

    //intList sert a stocker les donnees de ChoiceBox
    ObservableList<Integer> intList;

    {
        Integer[] inta = new Integer[16];
        int k = 1;
        for (int i = 0; i < inta.length; i++) {
            inta[i] = new Integer(k);
            k <<= 1;
        }
        intList = FXCollections.observableArrayList(inta);
    }
```

ChoiceBoxView avec un ChangeListener (suite)

```
    //constructeur
public ChoiceBoxView(Model model) {

 this.model = model;

 setItems(intList);
 // attacher un ChangeListener
 valueProperty().addListener(new ChangeListener<Object>() {

 @Override
 public void changed(ObservableValue<? extends Object> ov,
         Object old_val, Object new_val) {
      model.setEntier(((Integer) new_val).intValue());
  }
});
```

Le modèle de ChoiceBox possède aussi la propriété `selectedIndex`, donc une autre possibilité consiste à attacher un ChangeListener au modèle (voir le tutorial sur ChoiceBox) :

```
getSelectionModel().selectedIndexProperty()
                   .addListener(new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue<? extends Number> ov,
            Number old_val, Number new_val) {

        model.setEntier(intList
                .get(new_val.intValue()).intValue());
    }
});
```

Gestion d'evenements

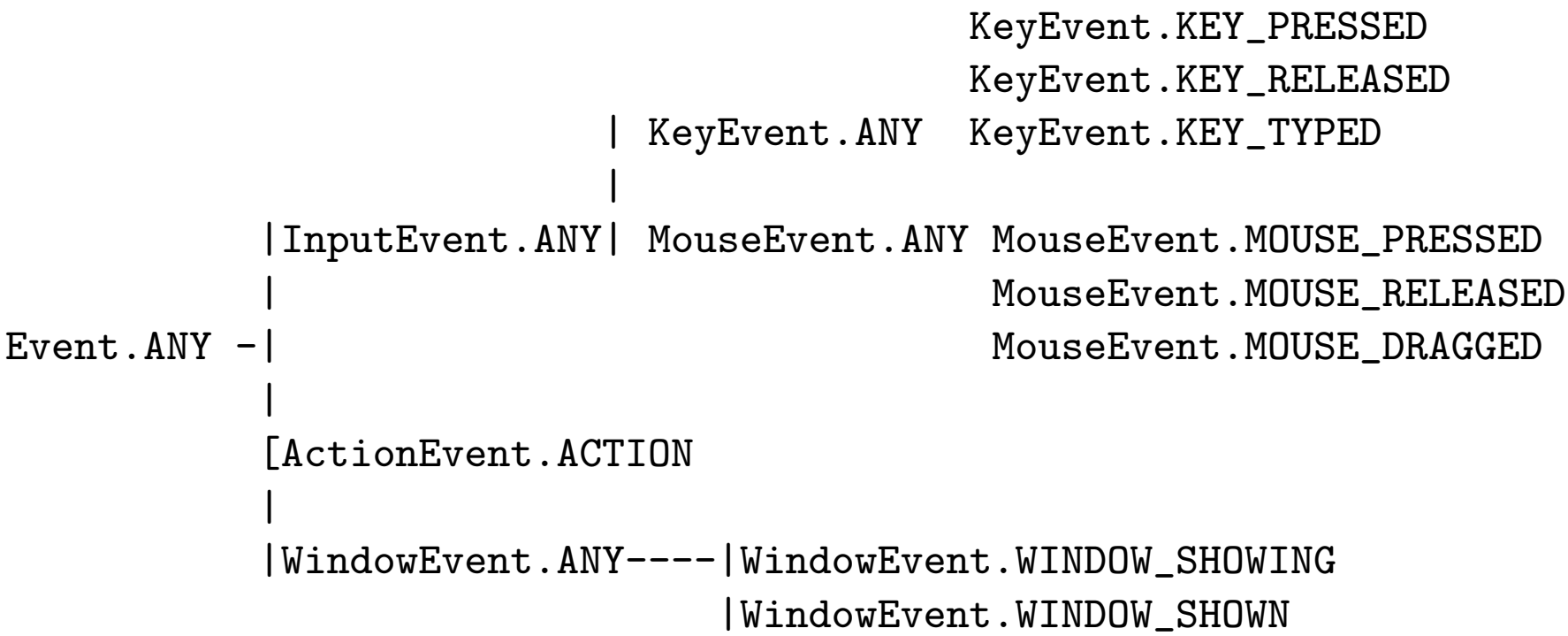
Évènement correspond souvent à une action de l'utilisateur : click de souris, taper sur le clavier, un geste sur l'écran tactile.

Un évènement est une instance de `javafx.event.Event`.

Les types évènements sont regroupés dans de classes d'évènements, dans chaque classe il y a plusieurs types d'évènements.

Exemple : la classe `KeyEvent` contient les types : `KEY_PRESSED`, `KEY_RELEASED`, `KEY_TYPED`.

Hiérarchie (partielle) d'évènements



Propagation d'évènements

Click sur un rectangle :

$$\text{Stage} \rightarrow \text{Scene} \rightarrow \text{BorderPane} \rightarrow \text{Pane} \rightarrow \text{Rectangle}$$

L'évènement se propage d'abord depuis `Stage` jusqu'au `Rectangle` (Event Capturing Phase) et ensuite dans le sens inverse depuis Rectangle jusqu'à `Stage` (Event Bubbling Phase).

A chaque noeud (Node) nous pouvons associé des `EventFilters` et des `EventHandlers` appelés pour traiter l'évènement.

Quelle est la différence entre un `EventFilter` et un `EventHandler` ?

Un `EventFilter` est exécuté pendant Event Capturing Phase.

Un `EventHandler` est exécuté pendant Event Bubbling Phase.

Un noeud peut enregistrer plusieurs filtres, l'ordre d'exécution est basé sur la hiérarchie d'évènement, d'abord les filtres plus spécifiques ensuite plus généraux (`MouseEvent.MOUSE_PRESSED` avant `MouseEvent.ANY`).

Un noeud peut enregistrer plusieurs handlers, l'ordre d'exécution est basé sur la hiérarchie d'évènement, d'abord les filtres plus spécifiques ensuite plus généraux (`KeyEvent.KEY_TAPED` avant `KeyEvent.ANY`).

Arrêter la propagation d'un évènement

La propagation d'un évènement peut-être arrêtée en exécutant la méthode

```
event.consume()
```

Un évènement consumé par un EventFilter ne passe pas dans la classe enfant, un évènement consumé par un EventHandler ne passe pas dans la classe parent.

Si un noeud possède plusieurs EventFilters qui correspondent à un évènement alors même si cet évènement est consumé par un EventFilter tous les filtres seront quand même exécutés.
Même chose pour EventHandlers.

Construire et enregistrer EventFilters

```
node.addEventFilter(MouseEvent.MOUSE_CLICKED,
  new EventHandler<MouseEvent>(){
      public void handle(MouseEvent event){
          //implementer le filtre
          //event.consume();
      }
  });
```

Noter que `addEventFilter` a deux paramètres :

- un EventType et
- un EventHandler.

Construire et enregistrer EventHandlers

```
node.addEventHadler(KeyEvent.KEY_TAPED,
  new EventHandler<KeyEvent>(){
      public void handle(KeyEvent event){
          //implementer le filtre
          //event.consume();
      }
  });
```

Noter que `addEventHandler` a les mêmes paramètres que `addEventFilter`.

Enregistrer un seul EventHandler

La classe `Node` (donc toutes les classes dérivées de Node) fournit des méthodes `setOn`*NomHandler* de commodité qui permettent d'enregistrer un seul handler, par exemple

```
node.setOnMousePressed(new EventHandler<MouseEvent>() {
    @Override
    public void handle(final MouseEvent event) {

    }
});
```

Ces méthodes correspondent aux propriétés de la classe `Node` (par exemple `MousePressedProperty`, `MouseDraggedProperty` etc.). L'avantage de ces méthodes par rapport à ma méthode `addEventHandler()` c'est la simplicité, un seul argument à la place de deux arguments.

Si vous voulez supprimer un EventHandler installé par une de méthodes `setOn` il suffit de passer l'argument null.

La liste d'enfant d'un noeud

Si on a un noeud (par exemple `Pane`) de type Parent on lui ajoute un enfant par

```
pane.getChildren().add( enfant );
```

Chaque parent possède une liste des enfants et `getChildren()` retourne cette liste. La méthode `add()` ajoute `enfant` à la fin de la liste.

Quand le parent est rendu sur l'écran les enfants sont rendus dans l'ordre de leur apparition sur la liste des enfants, le premier celui en tête de la liste jusqu'au dernier, celui à la fin de la liste.