Interfaces Java

Événements et liaison

Événements et liaisons

Contenu :

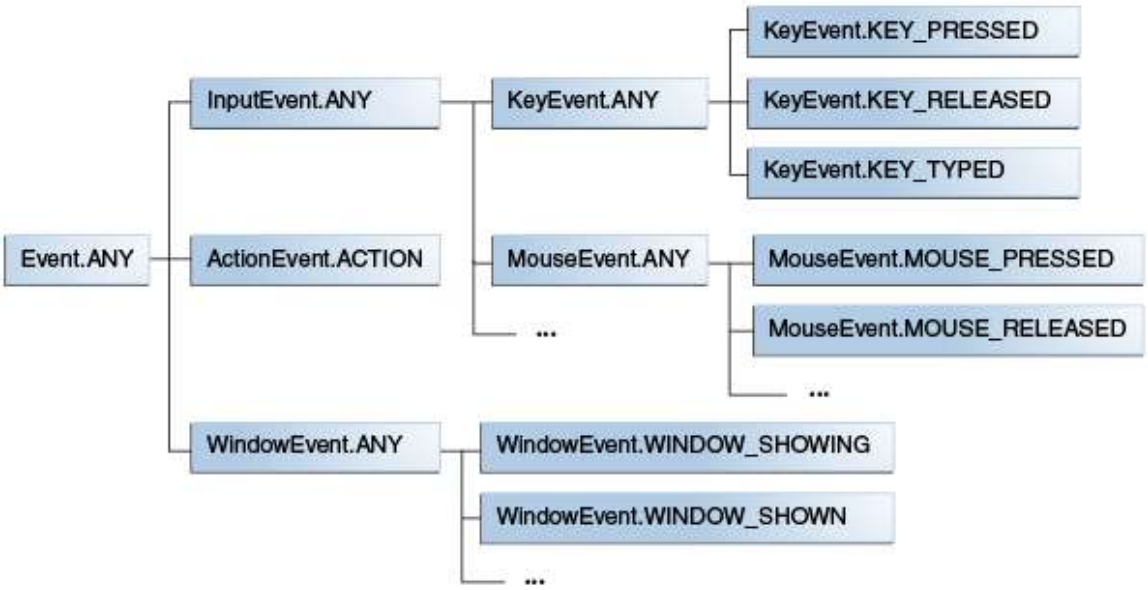
- Introduction aux événements
- Méthodes de commodité
- Écouteurs
- La liaison c'est quoi ?

Introduction aux événements

Les nœuds dans une application JavaFX communiquent grâce aux événements. Les événements permettent d'associer un traitement à une action.

Les événements sont utilisés pour avertir votre application d'une action initiée par l'utilisateur et pour permettre à l'application de répondre adéquatement. La plate-forme JavaFX fournit la structure pour capturer un événement, l'acheminer à sa cible, et pour permettre à l'application de gérer l'événement en fonction des besoins.

Un événement représente une occurrence de quelque chose d'intérêt pour l'application, par exemple une souris déplacée ou une touche du clavier enfoncée. En JavaFX, un événement est une instance de la classe javafx.event.Event ou de ses sous-classes. JavaFX fournit plusieurs événements notamment : DragEvent, KeyEvent, MouseEvent, ScrollEvent, etc. Vous pouvez aussi définir votre propre événement en héritant de la classe Event. Voici la hiérarchie des événements.



Tout événement produit l'information suivante :

- Le type de l'événement (keyPressed, keyReleased, MouseClicked, etc.)
- La cible « target », le nœud à l'origine de l'événement (bouton, contrôle, son, etc.).

Processus de transmission de l'événement

Voici les étapes de la transmission d'un événement :

1. La sélection de la cible « target »
2. Construction de la route de transmission
3. Capture de l'événement
4. Remonter « bubbling » l'événement (propagation)

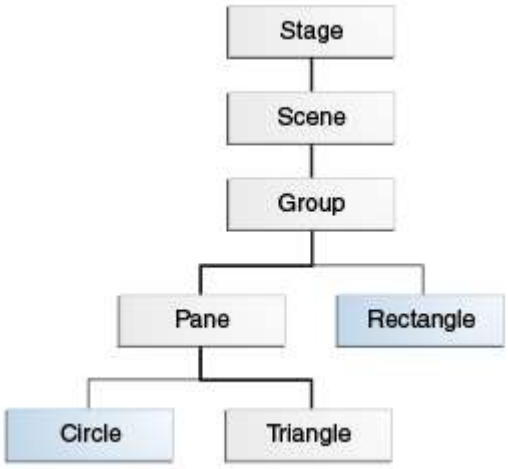


Figure 1 ***En gris, route suivie si l'événement consiste à cliquer sur le nœud Triangle.***

Dans la phase de **capture**, l'événement est envoyé par le nœud racine de votre application et transmis par la route de transmission vers le nœud cible en vue du traitement de l'événement. À chaque nœud de la route, si un événement filtre (event filter) est enregistré, on applique le traitement nécessaire. On peut consommer l'événement avec l'instruction consume(), lorsqu'un événement est consommé, son traitement est terminé. Par exemple, si dans le nœud **Pane** dans le diagramme ci-dessus, on rencontre l'instruction consume(), le nœud Triangle ne recevra jamais l'événement.

Consommer l'événement dans un gestionnaire d'événements arrête tout autre traitement de l'événement par les gestionnaires parents sur la chaîne d'expédition de l'événement.

Puis, on parcourt la route en sens inverse jusqu'à la racine, à chaque nœud, on vérifie s'il y a un gestionnaire d'événement associé et on applique le traitement nécessaire. Si l'événement n'est pas consommé avant que l'on soit revenu à la racine, l'événement et le traitement sont terminés au nœud Racine.

En résumé :

Un filtre « **EventFilter** » a les caractéristiques suivantes :

- Exécuté durant la phase de capture (étape 3)
- Peut avoir plusieurs filtres par nœud
- On enregistre un filtre ainsi :

```
noeud.addEventFilter(MouseEvent.MOUSE_PRESSED,
                    new EventHandler<MouseEvent>()
                    {       @Override
                            public void handle(MouseEvent event) {...}
                    });
```

Un événement « **EventHandler** » a les caractéristiques suivantes :

- Exécuté durant la phase de propagation « bubbling » (étape 4) donc après un filtre
- Peut avoir plusieurs gestionnaires d'événement « handler » par nœud
- On enregistre un gestionnaire d'événement ainsi :

```
noeud.addEventHandler(MouseEvent.MOUSE_PRESSED,
                    new EventHandler<MouseEvent>()
                    {       @Override
                            public void handle(MouseEvent event) {...}
                    });
```

Dans notre cours, nous allons nous intéresser aux **événements** et non aux filtres et nous allons essentiellement nous intéresser aux méthodes de commodités sauf pour les listes où nous devrons travailler avec des écouteurs « listener » particuliers.

Événements

Un événement représente une occurrence de quelque chose d'intérêt pour l'application, par exemple une souris déplacée ou une touche enfoncée. En JavaFX, un événement est une instance de la classe javafx.event.Event. JavaFX fournit plusieurs types événements : MouseEvent, DragEvent, KeyEvent, , ScrollEvent, et bien d'autres. Vous pouvez même définir votre propre événement en étendant la classe Event.

Dans notre cours nous allons explorer les événements de type ActionEvent, MouseEvent et KeyEvent.

Méthodes de commodité

Les méthodes de commodité sont une façon pratique que vous pouvez utiliser pour enregistrer des gestionnaires d'événements dans votre application JavaFX. Les méthodes de commodités sont disponibles pour créer et enregistrer des gestionnaires d'événements pour répondre aux événements de souris, les événements de clavier, des événements d'action, les événements glisser-déposer, les événements de la fenêtre, etc. Bref, les méthodes de commodité sont disponibles pour les événements courants qu'une application peut avoir besoin.

Une méthode de commodité possède cette syntaxe :

```
setOnEvent-type(EventHandler<? super event-class> value)
```

Événement Action

ActionEvent est le plus facile et le plus commun des événements.

Il s'agit généralement de l'action la plus courante associée à un nœud. Par exemple, l'événement action d'un bouton est le « clic », celui d'une zone de texte est la touche « Entrée » du clavier, etc.

Pour utiliser un événement Action il faut :

- Utiliser la méthode de commodité setOnAction
- Associer le gestionnaire ActionEvent.
- Redéfinir la méthode `handle(ActionEvent ev)`. EventHandler est une interface qui nécessite de redéfinir la méthode handle. C'est dans cette méthode que nous plaçons le traitement à faire suite à l'événement.

Par exemple :

```
bouton.setOnAction(new EventHandler<ActionEvent>(){
          public void handle(ActionEvent event) {
                  System.out.println("Hello World");
          }
  });
```

Est équivalent à :

```
bouton.setOnAction(new ActionBouton());

private class ActionBouton implements EventHandler<ActionEvent> {
        public void handle(ActionEvent event) {
                System.out.println("Hello World");
        }
});
```

Remarquez où et comment le gestionnaire d'événement « EventHandler » est créé et où est définie sa méthode « handle ». Il s'agit ici d'une classe anonyme où on redéfinit immédiatement la méthode sans se créer d'objet spécifique, new EventHandler n'a pas de nom, il est anonyme et on ne peut y référer par la suite. Chez Oracle, c'est ainsi qu'on définit un événement, car cela permettra plus tard de travailler avec des expressions lambdas. Mais, dans notre cours, nous allons préférer créer une classe concrète pour créer un événement et lui définir l'action à prendre.

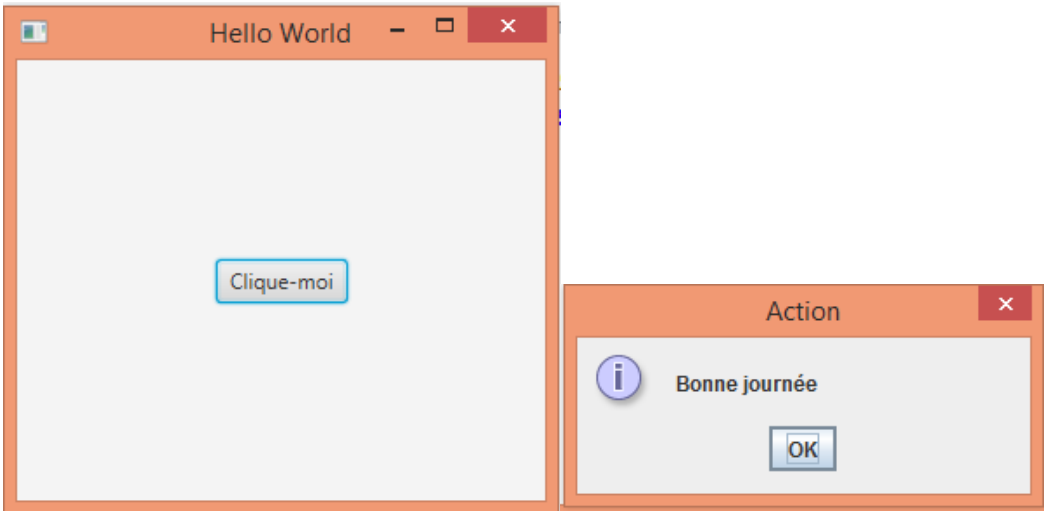
Par exemple :

```
btn.setOnAction(new ActionBouton());
```

et la classe ActionBouton est définie dans une autre classe ou dans une classe interne.

```
public class ActionBouton implements EventHandler<ActionEvent>
{
        @Override
        public void handle(ActionEvent event)
        {
            if (event.getSource() == btn)
            {
                JOptionPane.showMessageDialog(null,
                    "Action",  "Bonne journée", JOptionPane.INFORMATION_MESSAGE);
            }
        }
}
```

Exemple : Action




```
import javax.swing.JOptionPane;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Action extends Application
{
       private BorderPane root;
       private Scene scene;
       private Button btn;

       @Override
       public void start(Stage primaryStage)
       {
```

```
            btn = new Button();
            btn.setLayoutX(100);
            btn.setLayoutY(80);
            btn.setText("Clique-moi");
            btn.setOnAction(new ActionBouton());
            root = new BorderPane();
            root.setCenter(btn);
            scene = new Scene(root, 300, 250);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Hello World");
            primaryStage.show();
      }

      private class ActionBouton implements EventHandler<ActionEvent>
      {
            public void handle(ActionEvent event)
            {
                  if (event.getSource() == btn)
                  {
                     JOptionPane.showMessageDialog(null,
                        "Bonne journée", "Action", JOptionPane.INFORMATION_MESSAGE);

                  }
            }
      }

      public static void main(String[] args)
      {
            Application.launch(args);
      }
}
```

Événements Souris

Les méthodes de commodité pour l'enregistrement des gestionnaires d'événements pour la souris comprennent de nombreuses possibilités toutes issues de la classe MouseEvent :

setOnMouseClicked	Clic de souris
setOnMouseEntered	À l'entrée de la souris dans la zone
setOnMouseExited	À la sortie de la souris de la zone
setOnMousePressed	Sur pression d'un bouton de la souris.
setOnMouseReleased	Sur relâchement de la souris
setOnMouseMoved	Sur déplacement de la souris
setOnMouseDragged	Sur glissement « drag » de la souris

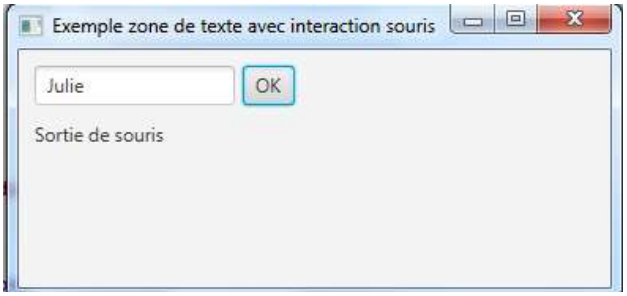
Chaque événement s'utilise de la même façon, on accroche un écouteur au nœud désiré et on définit l'action à prendre quand l'événement survient :

btn.setOnMousePressed(new EvenementSouris());

et la classe EvenementSouris est définie dans une autre classe ou dans une classe interne.

public class EvenementSouris implements EventHandler<**MouseEvent**> {
@Override
public void handle(MouseEvent event) {

Exemple : Souris




```
public class Souris extends Application
{
	private Scene scene;
	private GridPane grid;
	private TextField prenom;
	private Label commentaire;
	private Button ok;

	@Override
	public void start(Stage stage)
	{
		prenom = new TextField();
		prenom.setPromptText("Entrez votre prénom.");
		prenom.getText();
		commentaire = new Label();
		ok = new Button("OK");

		grid = new GridPane();
		grid.setPadding(new Insets(10, 10, 10, 10));
		grid.setVgap(5);
		grid.setHgap(5);
		GridPane.setConstraints(prenom, 0, 0);
		GridPane.setConstraints(commentaire, 0, 2);
		GridPane.setConstraints(ok, 1, 0);
		grid.getChildren().add(prenom);
		grid.getChildren().add(commentaire);
		grid.getChildren().add(ok);

		ajouterEcouteurs();
		scene = new Scene(grid, 375, 150);
		stage.setScene(scene);
		stage.setTitle("Exemple zone de texte avec interaction souris");
		stage.show();
	}

	private void ajouterEcouteurs()
	{
		ok.setOnAction(new ActionBouton());
		prenom.setOnMouseClicked(new EvenementSouris());
		prenom.setOnMouseEntered(new EvenementSouris());
		prenom.setOnMouseExited(new EvenementSouris());
		prenom.setOnMousePressed(new EvenementSouris());
	}

	private class ActionBouton implements EventHandler<ActionEvent>
```

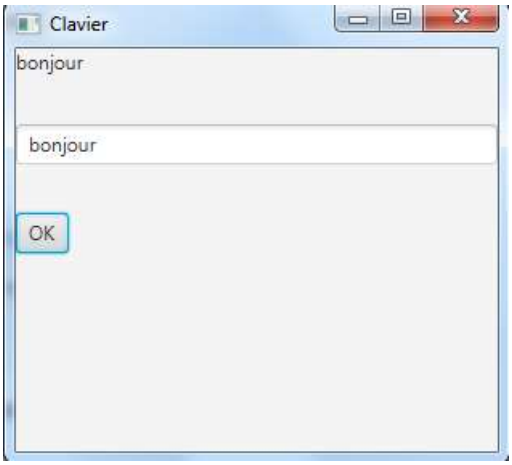
```
	{
		public void handle(ActionEvent event)
		{
			if (event.getSource() == ok)
			{
				commentaire.setText("Bonne journée " + prenom.getText());
			}
		}
	}

	private class EvenementSouris implements EventHandler<MouseEvent>
	{
		@Override
		public void handle(MouseEvent event)
		{
			if (event.getEventType() == MouseEvent.MOUSE_CLICKED)
			{
				commentaire.setText("Clic bouton");

			}
			else if (event.getEventType() == MouseEvent.MOUSE_ENTERED)
			{
				commentaire.setText("Entrée souris à la coordonnée x "
						+ event.getX());
			}
			else if (event.getEventType() == MouseEvent.MOUSE_EXITED)
			{
				commentaire.setText("Sortie de souris ");
			}
			else if (event.getEventType() == MouseEvent.MOUSE_PRESSED)
			{
				// ici la réponse sera PRIMARY (gauche) MIDDLE, SECONDARY
				// (droite) ou NONE
				System.out.println("Souris pressée avec le bouton "
						+ event.getButton());
			}
		}
	}

	public static void main(String[] args)
	{
		launch(args);
	}
}
```

Événement clavier

Les méthodes écouteurs pour l'enregistrement des gestionnaires d'événements pour le clavier comprennent setOnKeyPressed, setOnKeyTyped et setOnKeyReleased. On comprend facilement le premier et le dernier, pour le KeyTyped, celui-ci se contente d'écouter les touches du clavier central et non celles des touches de direction et des touches de fonction.

Un événement clavier permet de récupérer la touche qui a été appuyée grâce à la méthode **getCode** et de connaître si une touche spéciale a été appuyée grâce aux méthodes isAltDown(), isControlDown, etc. Reportez-vous à l'API pour plus de détails.

Exemple : Clavier




```
/**
 * Événement clavier qui n'enregistre que les lettres alphabétiques
 */
public class Clavier extends Application
{
	private VBox root;
	private Scene scene;
	private Button btn;
	private TextField texte;
	private String lettres;
	private Label label;

	@Override
	public void start(Stage primaryStage)
	{
		lettres = "";
		label = new Label();

		texte = new TextField();
		texte.setEditable(false);
		texte.setOnKeyPressed(new EcouteurTexte());

		btn = new Button();
		btn.setText("OK");
		btn.setOnAction(new ActionBouton());

		root = new VBox();
		root.setSpacing(30);
		root.getChildren().addAll(label, texte, btn);
		scene = new Scene(root, 300, 250);
		primaryStage.setScene(scene);
		primaryStage.setTitle("Clavier");
		primaryStage.show();
	}

	private class EcouteurTexte implements EventHandler<KeyEvent>
	{
		@Override
		public void handle(KeyEvent event)
		{
```

```
                    if (event.getSource() == texte)
                    {
                          KeyCode carac = event.getCode();
                          System.out.println(event.getCode());
                          if (carac.isLetterKey() || carac.isWhitespaceKey())
                          {
                                lettres += event.getText();
                                System.out.println(event.getText());
                          }
                          texte.setText(lettres);
                    }
             }
      }

      private class ActionBouton implements EventHandler<ActionEvent>
      {

             @Override
             public void handle(ActionEvent event)
             {
                    label.setText(lettres);
             }
      }

      public static void main(String[] args)
      {
             Application.launch(args);
      }


}
```

Il est important de prendre conscience que le programme fonctionne parce que le focus est sur la zone de texte, si ce n'était pas le cas, l'instruction requestFocus() permet d'amener le focus sur le nœud désiré.