Advanced Programming Methods

Iuliana Bocicor
maria.bocicor@ubbcluj.ro

Babes-Bolyai University

2023

Overview

Processing events

- Event delivery
- Event handling
- Convenience methods

Model-View-Controller

FXML

Events I

- The following slides are based on the documentation available at: JavaFX: Handling Events.
- Any user action generates an event, e.g.:
  - pressing or releasing of a key;
  - moving, clicking, releasing the mouse;
  - opening or closing a window;
  - scrolling;
- Any event in JavaFX is a subclass of javafx.event.Event.
- Any event has:
  - *a type*: an instance of the EventType class, e.g.:
    - KeyEvent.KEY_PRESSED
    - MouseEvent.MOUSE_RELEASED
    - WindowEvent.WINDOW_SHOWN

Events II

- *a source*: the origin of the event, considering the event location in the dispatch chain; the source can change as the event is passed in the chain.
- *a target*: the node on which the action occurred; this cannot be changed.
- Any class implementing the EventTarget interface can be the target of an event. E.g.: Window, Scene, Node.

The event delivery process

- There are several phases happening whenever an event is generated:
  1. Target selection
  2. Route construction
  3. Event capturing
  4. Event bubbling

Target selection

- The target of the action is determined according to a set of rules, e.g.:
    - For key events - the target is the node that has the focus.
    - For mouse events - the target is the node at the location of the cursor.
    - For continuous gesture events that are generated by a gesture on a touch screen - the target is the node at the centre point of all touches at the beginning of the gesture.
- If there is more than one node located at the cursor or touch location, the topmost is considered the target.

Route construction

Figure: Figure source: Sample user interface.

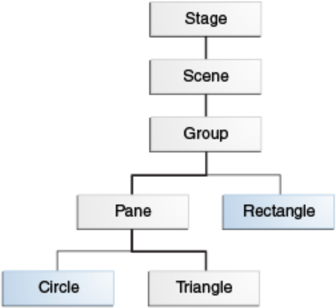

Figure: Figure source: Event dispatch chain.

- The default event route is the path from the stage to the involved node.

Event capturing phase

- In this phase the event moves down the path towards the target node.
- If along this path there is some filter that processes it, the filter is called and then the event continues its way.
- If there is a filter that consumes the event, then the event is no longer passed to the next node.
- Otherwise, the event continues to be passed down the path until it reaches it target, which will process it.

Event bubbling phase

- In this phase the event travels the other way around, from the target to the root node.
- If the target or any other node up the path has a handler for the event the handler is called and then the event continues to be passed to the next node.
- If any handler consumes the event, this will no longer continue its travel to the root node.
- If no handler consumes the event, this will eventually reach the root node and event processing is completed.

Event handling I

- This is achieved via filters and handlers, both of which implement the EventHandler interface.
- Filters and/or handlers have to be registered for a certain event to get the chance to process it.
- Filters are executed during the event capturing phase, while handlers during the event bubbling phase.
- Thus, the difference between filters and handlers is mainly when each one is executed.
- A filter for a parent node can provide common event processing for multiple child nodes.

Event handling II

- A node can register multiple filters.
- As an event passes through a node that has a registered filter for that event, the filter is executed. The event can then be consumed by the filter or it can continue to the next nodes down the path.
- As an event passes through a node that has a handler registered for that event, the handler is executed and the event continues to the next node up the path.
- A node can register multiple handlers.

Event handling III

- If an event is consumed by a filter or a handler, it will no longer continue to be passed up/down the path.
- Consuming an event by a filter means that no child node in the chain can act on the event. Consuming an event by a handler means that no parent node in the chain can act on it.
- An event can be consumed with the consume() method.
- The default handlers for the JavaFX UI controls typically consume most of the input events.

Event filters

- A filter can be used for more than one node and more than one event type.
- Through event filters parent nodes can provide common processing for child nodes.
- To process an event during the event capturing phase, a node must register an event filter.
- The code that is executed when the event is intercepted should be in the handle() method implementation. The method is defined in the EventHandler interface.
- The addEventFilter() method should be used to register an event filter.
- The removeEventFilter() method should be used to remove the filter, to avoid any future processing of the event.

Event handlers

- A handler can be used for more than one node and more than one event type.
- Through event handlers parent nodes can provide common processing for child nodes.
- To process an event during the event bubbling phase, a node must register an event handler.
- The code that is executed when the event is intercepted should be in the handle() method implementation. The method is defined in the EventHandler interface.
- The addEventHandler() method should be used to register an event handler.
- The removeEventHandler() method should be used to remove the handler, to avoid any future processing of the event.

Convenience methods I

- Some JavaFX classes define event handler properties whose values can be set through setter methods.
- These methods are known as *convenience methods*.
- This offers a way to easily register event handlers.
- Classes like Node, Scene, Window and their subclasses define such methods.
- A convenience method has the following format:

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

Convenience methods II

- where:
  - *Event-type* is directly related to the event type, e.g. for MOUSE_CLICKED events: setOnMouseClicked.
  - The method accepts an event handler for an event class (e.g. KeyEvent) or for any of its parent classes.
- For example:

```
setOnKeyTyped(
    EventHandler<? super KeyEvent> value)
```

Example

Example

Lecture7_demo1.

Model-View-Controller (MVC) I

Is an architectural pattern used to separate the application concerns.

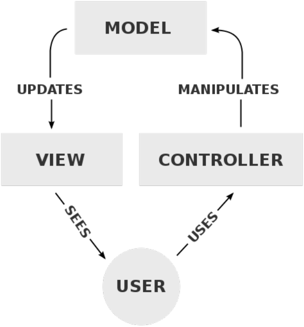

Figure: Figure source: Wikipedia

Model-View-Controller (MVC) II

Model

- Represents and manages the data of the application domain.
- Is responsible for:
  - fetching the data that is needed for view;
  - writing back any changes (requests which come from the controller).

Model-View-Controller (MVC) III

View

- Presents the data to the user.
- Even if we have a large dataset, only a limited amount of data is visible. That is the only data that is requested by the view.

Controller

- Mediates between the user and the view.
- Interprets user input and commands the model or the view to change as appropriate.
- Converts user actions (which come from the view) into requests to navigate or edit data.

FXML

- FXML is an XML-based declarative annotation language.
- It can be used to design GUIs, without the need for the application to be recompiled each time elements within it are modified.
- In this way a separation is made between the *presentation level* and the *logic level* of an application.
- JavaFX *Scene Builder* is a visual layout tool that allows quickly designing graphical user interfaces, without coding.
- Users can just drag and drop components and modify their properties and the FXML code is automatically generated.
- The generated FXML can then be combined with a Java project by binding the UI to the application's logic.

Programmatic and declarative I

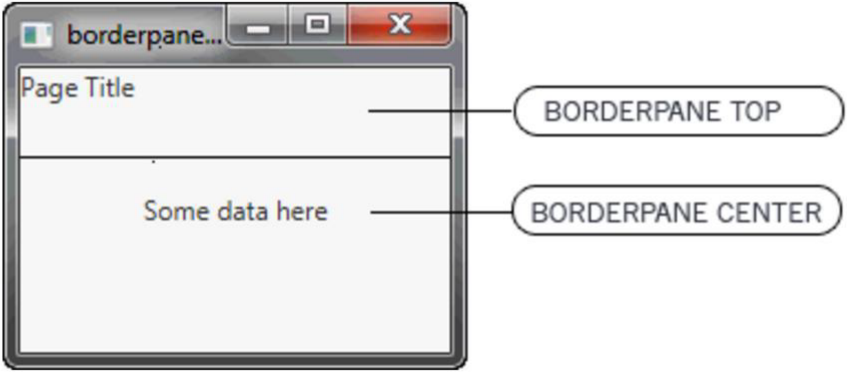

Figure: Figure source: MAP Lectures (Camelia Serban)

Programmatic and declarative II

Programmatic

```
BorderPane border = new BorderPane();
Label top = new Label(Label("Page Title"));
border.setTop(top);
Label center = new Label("Some data here");
border.setCenter(center);
```

Programmatic and declarative III

Declarative

```
<BorderPane>
    <top>
        <Label text="Page Title"/>
    </top>
    <center>
        <Label text="Some data here"/>
    </center>
</BorderPane>
```

Elements in FXML I

- The following slides are based on the documentation available at: Using FXML to Create a User Interface.
- To create the following interface, use Scene Builder. This can be downloaded from: this link.
- Please see the resulting *fxml* file in the example Lecture7_demo2.

Elements in FXML II

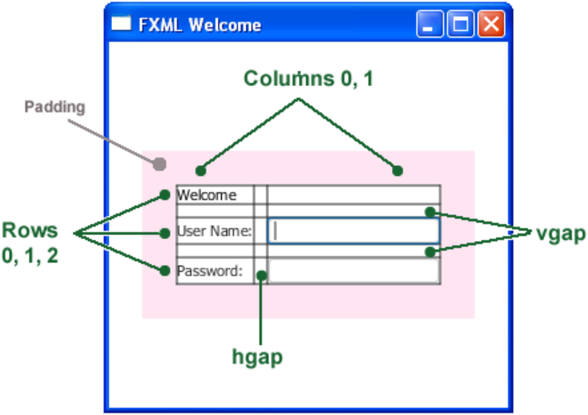

Figure: Figure source: Login form.

CSS

- A style can be applied using a *css* file.
- The file must be created and saved within the project.
- As an example, please see the *Login.css* file from Lecture7_demo2.

```
<GridPane hgap=”10.0” maxHeight=”-Infinity” maxWidth=”-Infinity”
minHeight=”-Infinity” minWidth=”-Infinity” prefHeight=”400.0”
prefWidth=”600.0” stylesheets=”@Login.css” vgap=”10.0”
xmlns:fx=”http://javafx.com/fxml/1”
xmlns=”http://javafx.com/javafx/11.0.1”
fx:controller=”Controller”>
```

FXML Loader

```
Parent root = FXMLLoader.load(getClass().getResource(
                            "sample.fxml"));
stage.setScene(new Scene(root, 550, 350));
```

or

```
FXMLLoader loader = new FXMLLoader();
loader.setLocation(getClass().getResource("sample.fxml"));
GridPane root = (GridPane) loader.load();
stage.setScene(new Scene(root, 550, 350));
```

FXML Controller I

```
<GridPane hgap=”10.0” maxHeight=”-Infinity” maxWidth=”-Infinity”
minHeight=”-Infinity” minWidth=”-Infinity” prefHeight=”400.0”
prefWidth=”600.0” stylesheets=”@Login.css” vgap=”10.0”
xmlns:fx=”http://javafx.com/fxml/1”
xmlns=”http://javafx.com/javafx/11.0.1”
fx:controller=”sample.Controller” >
```

FXML Controller II

- The name of the controller class can be specified in the *fxml* file, as seen above.
- In this case, the class is called Controller.
- Alternatively, the controller object can be created and set to the FXMLLoader.
- The Controller class will handle user interaction (event handling).
- The UI controls will be used in the Controller class via their fx:id.
  - <TextField fx:id="userInput" promptText="username" GridPane.columnIndex="1" GridPane.rowIndex="1">

FXML Controller III

```
public class Controller {
    @FXML private TextField userInput;
    @FXML private PasswordField passwordInput;
    @FXML private Button signInButton;

    // ...
}
```

FXML Controller IV

Handling events in the Controller

- For each control we can specify various handlers, for various actions.
- The handlers must then be implemented in the Controller class.

```
<Button fx:id=”signInButton” onAction=”handleSignInButtonClick”
text=”Sign in” textAlignment=”CENTER” GridPane.columnIndex=”1”
GridPane.rowIndex=”3”>
```

FXML Controller V

```
public class Controller {
    private static String USER_NAME = "map_user";
    private static String PASSWORD = "map_pass";

    @FXML private TextField userInput;
    @FXML private PasswordField passwordInput;
    @FXML private Button signInButton;

    @FXML
    private void handleSignInButtonClick(ActionEvent e)
    {
        if (this.userInput.getText().equals(USER_NAME) &&
        this.passwordInput.getText().equals(PASSWORD))
        {
            Alert alert = new Alert(
```

FXML Controller VI

```
                        Alert.AlertType.INFORMATION);
        alert.setTitle("All is well");
        alert.setHeaderText(
                  "Great! User and password validated.");
        alert.showAndWait();
    }
    else
    {
        // ERROR!
    }
}
```

Summary I

- User actions generate events, some of which must be handled.
- In JavaFX this can be achieved using event filters and/or event handlers (both implement the EventHandler interface.
- These should be created and registered to their corresponding nodes.
- Convenience methods offer an easy way to handle events. These are defined in classes Node, Scene, Window and their subclasses.
- MVC is an architectural pattern used to separate the application concerns.
- The *model* manages the data. The *view* presents the data. The *controller* mediates between the model and the view.

Summary II

- FXML s an XML-based declarative annotation language. It provides a way to separate the *presentation level* and *the logic level* of an application. It facilitates the implementation of MVC.
- *Next week:*
  - Java introspection and reflection.
  - Concurrency.