CSCI 2120:

Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX Core: FXML

Overview

Introduction

- ***JavaFX FXML*** is an XML format that enables you to compose JavaFX GUIs in a fashion similar to how you compose web GUIs in HTML.

- ***FXML*** thus enables you to separate your JavaFX layout code from the rest of your application code.

- This cleans up both the layout code and the rest of the application code.

- ***FXML*** can be used both to compose the layout of a whole application GUI, or just part of an application GUI, *e.g. the layout of one part of a form, tab, dialog etc.*

JavaFX FXML Example

The best way to learn about JavaFX FXML is to see an FXML example. Below is an example that composes a simple JavaFX GUI:

```
<? xml version="1.0" encoding="UTF-8" ?>
<? import javafx.scene.layout.VBox    ?>
<? import javafx.scene.control.Label  ?>

<VBox>
    <children>
        <Label text="Hello world FXML"/>
    </children>
</VBox>
```

Code Explanation:

This example defines a VBox containing a single Label as child element. The VBox component is a JavaFX layout component. The Label just shows a text in the GUI.

The first line in the ***FXML*** document is the standard first line of XML documents.

The following two lines are `import` statements. In ***FXML*** you need to import the classes you want to use. Both JavaFX classes and core Java classes used in ***FXML*** must be imported.

After the import statements you have the actual composition of the GUI. A VBox component is declared, and inside its children property is declared a single Label component. The result is that the Label instance will be added to the children property of the VBox instance.

Loading an FXML File

In order to load an ***FXML*** file and create the JavaFX GUI components, use the FXMLLoader `javafx.fxml.FXMLLoader` class.

Here is a full JavaFX ***FXML*** loading example which loads an FXML file and returns the JavaFX GUI component declared in it:

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.net.URL;

public class FXMLExample extends Application{
   public static void main(String[] args) {
       launch(args);
   }
   @Override
   public void start(Stage primaryStage) throws Exception {
       FXMLLoader loader = new FXMLLoader();
       loader.setLocation(new URL("./data/hello-world.fxml"));
       VBox vbox = loader.<VBox>load();
       Scene scene = new Scene(vbox);
       primaryStage.setScene(scene);
       primaryStage.show();
   }
}
```

Code Explanation:

For this example to work, the FXML file must be located at `./data/hello-world.fxml.`

As you can see, the location of the file is set via the `setLocation()` method.

The root GUI component (the VBox object) is obtained via the `load()` method.

Importing Classes in FXML

In order to use a Java class in ***FXML***, whether a JavaFX GUI component or a regular Java class, the class must be imported in the FXML file.

FXML import statements look like this:

```
<? import javafx.scene.layout.VBox ?>
```

This FXML import statement imports the class `javafx.scene.layout.VBox` .

Creating Objects in FXML

FXML can create both JavaFX GUI objects as well as non-JavaFX objects (POJOs).

There are several ways to create objects in FXML. In the following sections we will see what these options are.

1. Creating Objects via **FXML Elements** and **No-arg Constructors**
2. Creating Objects via ***valueOf()*** Method
3. Creating Objects via **Factory Methods**

Creating Objects - FXML Elements & No-arg Constructors

The easiest way to create objects in FXML is via an FXML element in an FXML file. The element names used in FXML are the same names as the Java class names without the package names. Once you have imported a class via an FXML import statement, you can use its name as an FXML element name.

In the following example the element names VBox and Label are valid because these two classes are declared with import statements earlier in the FXML file:

```xml
<? xml version="1.0" encoding="UTF-8" ?>
<? import javafx.scene.layout.VBox ?>
<? import javafx.scene.control.Label ?>

<VBox>
    <children>
        <Label text="Hello world FXML"/>
    </children>
</VBox>
```

Code Explanation:

To create objects using FXML elements like this requires that the class of the created object has a no-argument constructor.

Creating Objects - valueOf() Method

Another way to create objects in FXML is to call a static `valueOf()` method in the class you want to create the object of. The way to create objects via a `valueOf()` method is to insert a value attribute in the FXML element.

Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import dev.teddy.javafx.MyClass?>

<MyClass value="The Value"/>
```

Here is the corresponding `MyClass` code for this to work:

```
public class MyClass {
    public static MyClass valueOf(String value) {
        return new MyClass(value);
    }

    private String value;

    public MyClass(String value) {
        this.value = value;
    }
}
```

Code Explanation:

Notice the static `valueOf()` method which takes a **Java String** as parameter. This method is called by the `FXMLLoader` when it sees the `MyClass` element in the FXML file. The object returned by the `valueOf()` method is what is inserted into the GUI composed in the FXML file. The above FXML doesn't contain any other elements than the `MyClass` element, but it could.

Keep in mind that whatever object is returned by the `valueOf()` method will be used in the object graph (composed GUI). If the object returned is not an instance of the class containing the `valueOf()` method, but an instance of some other class, then that object will still be used in the object graph. The element name is used only to find the class containing the `valueOf()` method (when the FXML element contains a `value` attribute).

Creating Objects - Factory Methods

In a sense, a `valueOf()` method is also a factory method that creates objects based on a String parameter. But - you can also get the `FXMLLoader` to call other factory methods than a `valueOf()` method.

To call another factory method to create an object, you need to insert an `fx:factory` attribute. The value of the `fx:factory` attribute should be the name of the factory method to call.

Here is an example:

```
<? xml version="1.0" encoding="UTF-8" ?>
<? import dev.teddy.javafx.MyClass ?>

<MyClass fx:factory="instance"/>
```

The `MyClass` class should look like this for the above FXML to work:

```
public class MyClass {
   public static MyClass instance() {
       return new MyClass();
   }
}
```

Code Explanation:

Notice the `instance()` method. This method is referenced from the `fx:factory` attribute in the FXML snippet above.

Note, that the factory method must be a no-argument method to call it from a `fx:factory` attribute.

Properties in FXML

Some JavaFX objects have properties. In fact, most of them do. You can set the values of properties in two ways. The first way is to use an XML attribute to set the property value. The second way is to use a nested XML element to set the property value.

To see how to set properties in FXML elements,

Let's look at an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<? import javafx.scene.layout.VBox ?>
<? import javafx.scene.control.Label ?>

<VBox spacing="20">
   <children>
       <Label text="Line 1"/>
       <Label text="Line 2"/>
   </children>
</VBox>
```

Code Explanation:

This example shows 3 property examples. The first example is the `spacing` attribute in the `VBox` element. The value set in the `spacing` attribute is passed as parameter to the `setSpacing()` method of the `VBox` object created based on the `VBox` element.

The second example is the `children` element nested inside the `VBox` element. This element corresponds to the `getChildren()` method of the `VBox` class. The elements nested inside the `children` element will be converted to JavaFX components that are are added to the collection obtained from the `getChildren()` method of the `VBox` object represented by the parent `VBox` element.

The third example are the `text` attributes of the two `Label` elements nested inside the `children` . The values of the `text` attributes will be passed as parameters to the `setText()` property of the `Label` objects created by the `Label` elements.

Property Name Matching

FXML considers "properties" to be member variables accessed via getters and setters. E.g. `getText()` and `setText()` .

As you can see from the example in the previous section the property names of JavaFX classes are matched to the attribute and element names by:

- Remove any get/set in the property name.
- Convert first remaining character of property name to lowercase.

Thus, the getter method `getChildren` will first be reduced to `Children` and then to `children`.

Similarly, the setter method `setText` will be reduced to `Text` and then to `text.`

Default Properties

A JavaFX component can have a **default property**. That means, that if a FXML element contains children which are not nested inside a property element, then it is assumed that the children are belonging to the **default property**.

Let us look at an example. The `VBox` class has the `children` property as default property. That means that we can leave out the `children` element. Thus, this FXML on the *left* can be shortened to the FXML on the *right*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>

<VBox spacing="20">
   <children>
       <Label text="Line 1"/>
       <Label text="Line 2"/>
   </children>
</VBox>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>

<VBox spacing="20">
       <Label text="Line 1"/>
       <Label text="Line 2"/>
</VBox>
```

Code Explanation

The two `Label` elements are then assumed to belong to the default property of `VBox`, which is the `children` property. A default property is marked with the JavaFX annotation `@DefaultProperty(value="propertyName")` where the value is the name of the property that should be the default property. For instance, the `@DefaultProperty(value="children")` declaration would make the `children` property the default property.

FXML Namespace

FXML has a **namespace** you can set on the root element of your FXML files. The **FXML namespace** is needed for some FXML attributes like the `fx:id` attribute.

Setting the FXML namespace on the root element of an FXML file looks like this:

```
<? xml version="1.0" encoding="UTF-8" ?>
<? import javafx.scene.layout.VBox ?>

<VBox xmlns:fx="http://javafx.com/fxml">
</VBox>
```

Code Explanation

The **FXML namespace** *(xmlns)* is declared by the attribute declaration `xmlns:fx="http://javafx.com/fxml"`

FXML Element IDs

You can assign IDs to FXML elements. These IDs can be used to reference the FXML elements elsewhere in the FXML file. Specifying an ID for an FXML element is done via the `id` attribute from the FXML namespace.

Here is an example of specifying an ID for an FXML element:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>

<VBox xmlns:fx="http://javafx.com/fxml">
   <Label fx:id="label1" text="Line 1"/>
</VBox>
```

Code Explanation

Notice the attribute declaration `fx:id="label1"` in the `Label` element. This attribute declares the ID of that `Label` element. Now this specific `Label` element can be referenced via the ID `label1` elsewhere in the FXML document. For instance, this ID can be used to reference the FXML element from CSS.

FXML Event Handlers

It is possible to set event handlers on JavaFX objects from inside the FXML file that defines the JavaFX objects. You might prefer to set advanced event handlers from within Java code, but for simple event handlers setting them from within FXML might be fine.

In order to define an event handler you need to use a `script` element. Here is how an FXML script element looks:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>

<VBox xmlns:fx="http://javafx.com/fxml">
   <Label  fx:id="label1" text="Button not clicked"/>
   <Button fx:id="button1" text="Click me!" onAction="click()"/>

   <fx:script>
           function click() {
               label1.setText("Button clicked");
           }
   </fx:script>

</VBox>
```

Code Explanation:

An event listener is added to a JavaFX component in FXML. The `Button` element declares an event listener via its `onAction` attribute. The attribute value declares a call to the `click()` function which is defined in the `script` element.

Notice the reference of a JavaFX component via its ID from within the FXML. Inside the `click()` method in the `script` element, the `Label` element is referenced via its ID `label1`. The `onAction` event listener attribute corresponds to the `onAction` event of the `Button` component. You can set this event listener via Java code too, via the `Button setOnAction()` method.

You can set listeners for other events in FXML too, by matching their event listener methods from the corresponding JavaFX component with an FXML attribute, using the same name matching rules.

FXML CSS Styling

It is possible to style the JavaFX components declared inside an FXML file. You can do so by embedding a style element inside the FXML element.

Here is an example of CSS styling a JavaFX button in an FXML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox xmlns:fx="http://javafx.com/fxml">
   <Button text="Click me!"/ onAction="click()">
       <style>
           -fx-padding: 10;
           -fx-border-width: 3;
       </style>
   </Button>
</VBox>
```

Code Explanation:

This example sets the -fx-padding CSS property to 10, and the -fx-border-width property to 3.

Since the style element is nested inside the button element, these CSS styles will be applied to that button element.

FXML Controller Classes

You can set a controller class for an FXML document. An FXML controller class can bind the GUI components declared in the FXML file together, making the controller object act as a mediator (design pattern).

There are two ways to set a controller for an FXML file.

1. The first way to set a controller is to specify it inside the FXML file.
2. The second way is to set an instance of the controller class on the `FXMLLoader` instance used to load the FXML document.

Specifying Controller Class in FXML

The controller class is specified in the root element of the FXML file using the `fx:controller` attribute.

Here is an example of specifying a controller in FXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox xmlns:fx="http://javafx.com/fxml" fx:controller="dev.teddy.javafx.MyFxmlController" >
   <Button text="Click me!"/ onAction="click()">
   </Button>
</VBox>
```

Code Explanation:

Notice the `fx:controller` attribute in the root element (the `VBox` element). This attribute contains the name of the controller class. An instance of this class is created when the FXML file is loaded. For this to work, the controller class must have a no-argument constructor.

Setting a Controller Instance on the FXMLLoader

When setting a controller instance on an `FXMLLoader` you must first create an instance of the controller class, and then set that instance on the `FXMLLoader`.

Here is an example of setting a controller instance on an `FXMLLoader` instance:

```
MyFxmlController controller = new MyFxmlController();

FXMLLoader loader = new FXMLLoader();
loader.setController(controller);
```

Binding JavaFX Components to Controller Fields

You can bind the JavaFX components in the FXML file to fields in the controller class. To bind a JavaFX component to a field in the controller class, you need to give the FXML element for the JavaFX component an **`fx:id`** attribute which has the name of the controller field to bind it to as value.

Here is an example controller class:

```
public class MyFxmlController {
   public Label label1 = null;
}
```

And here is the FXML file with a `Label` element bound to the `label1` field of the controller class:

```
<VBox xmlns:fx="http://javafx.com/fxml" >
   <Label fx:id="label1" text="Line 1"/>
</VBox>
```

Notice how the value of the **`fx:id`** attribute has the value `label1` which is the same as the field name in the controller class to which it should be bound.

Referencing Methods in the Controller

It is possible to reference methods in the controller instance from FXML. For instance, you can bind the events of a JavaFX GUI component to methods of the controller.

Here is an example of binding an event of a JavaFX component to a method in the controller:

```
<VBox xmlns:fx="http://javafx.com/fxml" fx:controller="dev.teddy.javafx.MyFxmlController" spacing="20">
   <children>
       <Label fx:id="label1" text="Line 1"/>
       <Label fx:id="label2" text="Line 2"/>
       <Button fx:id="button1" text="Click me!" onAction="#buttonClicked"/>
   </children>
</VBox>
```

Referencing Methods in the Controller

This example binds the `onAction` event of the `Button` to the method `buttonClicked` in the controller class.

Here is how the controller class must look to enable the event binding:

```java
import javafx.event.Event;
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class MyFxmlController {
   @FXML
   public void buttonClicked(Event e){
       System.out.println("Button clicked");
   }
}
```

Notice the `@FXML` annotation above the `buttonClicked` method. This annotation marks the method as a target for binding for FXML. Notice also that the name `buttonClicked` is referenced in the FXML file.

Obtaining the Controller Instance From the FXMLLoader

Once the `FXMLLoader` instance has loaded the FXML document, you can obtain a reference to the controller instance via the `FXMLLoader getController()` method.

Here is an example:

```
MyFxmlController controllerRef = loader.getController();
```

END