howest
hogeschool

Object Oriented Architectures and Secure Development

User interfaces with JavaFX

Matthias Blomme

Mattias De Wael

Frédéric Vlummens

JavaFX: Hello world

Adding JavaFX dependencies to build.gradle

```gradle
plugins {
    id 'java'
    id 'org.openjfx.javafxplugin' version '0.0.10'
}



javafx {
    version = "18"
    modules = [ 'javafx.controls']
}
```

For the latest versions:
https://openjfx.io/openjfx-docs/#gradle

howest
hogeschool

The Application class

```java
public class HelloApp extends Application {

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        Label lbl = new Label("Hello World!");
        Scene scene = new Scene(lbl, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

howest
hogeschool

Do not launch your app from IntelliJ

This will cause an error:

> Task :HelloApp.main() FAILED
Error: JavaFX runtime components are missing, and are required to run this application

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':HelloApp.main()'.
> Process 'command '/Users/fredericvlummens/Library/Java/JavaVirtualMachines/azul-11.0.8/Contents/Home/bin/java'' finished with non-zero exit value 1

howest
hogeschool

Instead, configure your build.gradle

```
plugins {
    id 'java'
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.10'
}

application {
    mainClass = 'be.howest.ti.HelloApp'
}
```

Adding the application plugin, allows you specify a main class,
And it creates a run task. (double click it to start the app)
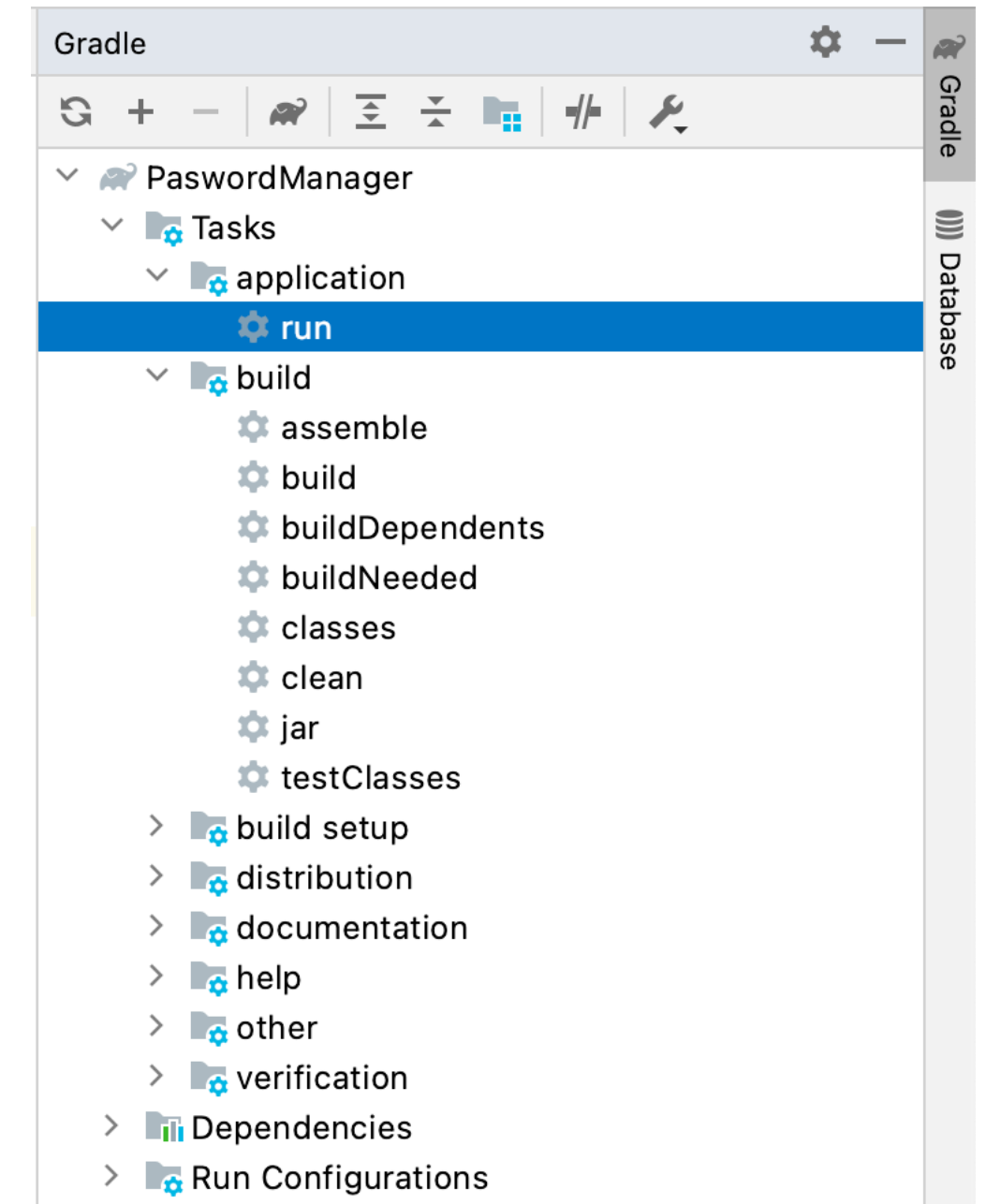
JavaFX: building an actual UI

Building an actual UI

- Not required to add all controls manually as in previous example

- We will be using FXML, an XML-based format to describe our UI

- Make sure to add the necessary dependency to **build.gradle**:

```
javafx {
    version = "18"
    modules = [ 'javafx.controls', 'javafx.fxml' ]
}
```

howest
hogeschool

The FXML file

- Stored in **/resources/fxml** (=convention followed in this course):

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity"
    minHeight="-Infinity" minWidth="-Infinity"
    prefHeight="100.0" prefWidth="200.0"
    xmlns="http://javafx.com/javafx/18">
  <Label>Hello</Label>
  <Label>World</Label>
  <Button>Do nothing</Button>
</VBox>
```
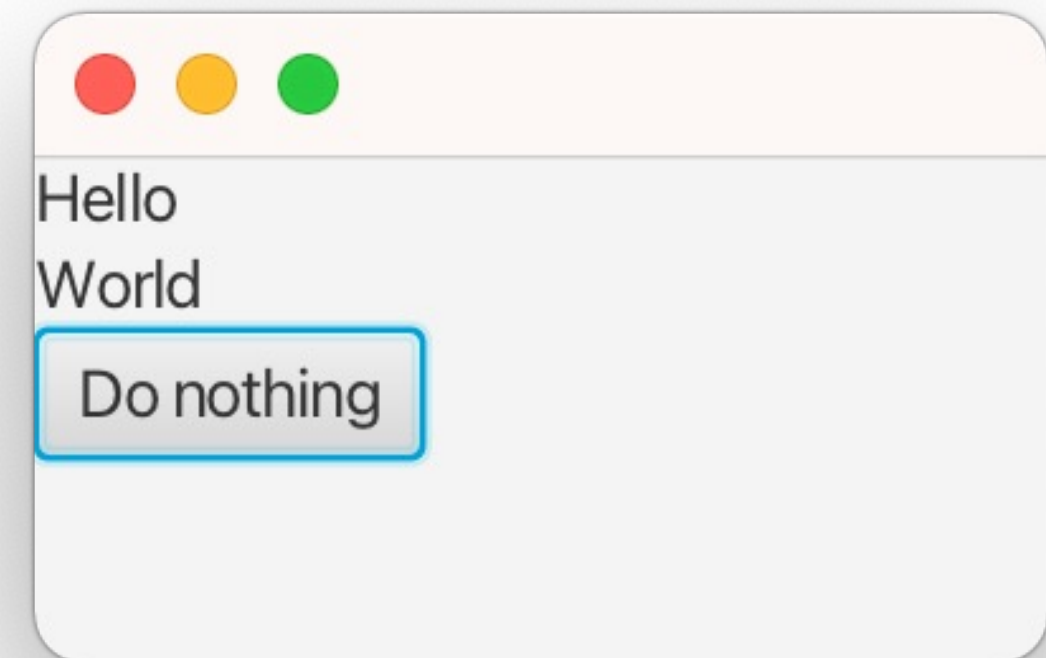
Load the FXML file in Application class

```java
public class FxApplication extends Application {

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


    @Override
    public void start(Stage primaryStage) throws IOException {
        Parent root = FXMLLoader.load(getClass().getResource("/fxml/demo.fxml"));
        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

howest
hogeschool

The FXML file, with controller

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="100.0" prefWidth="200.0"
    xmlns="http://javafx.com/javafx/18"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="be.howest.ti.shop.ui.fx.DemoController">
  <Label>Hello</Label>
  <Label fx:id="lblWord">World</Label>
  <Button onAction="#doSomething">Do something</Button>
</VBox>
```
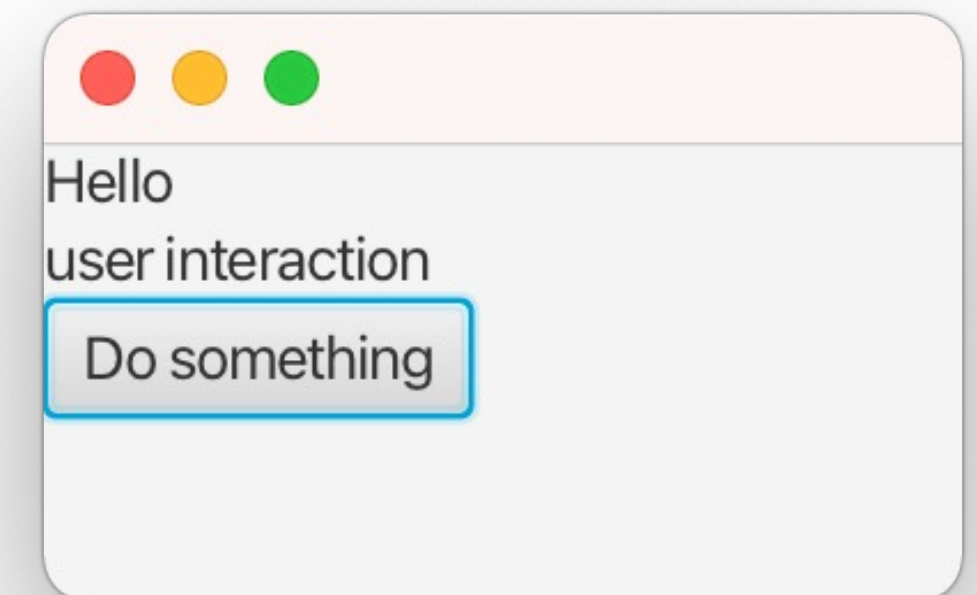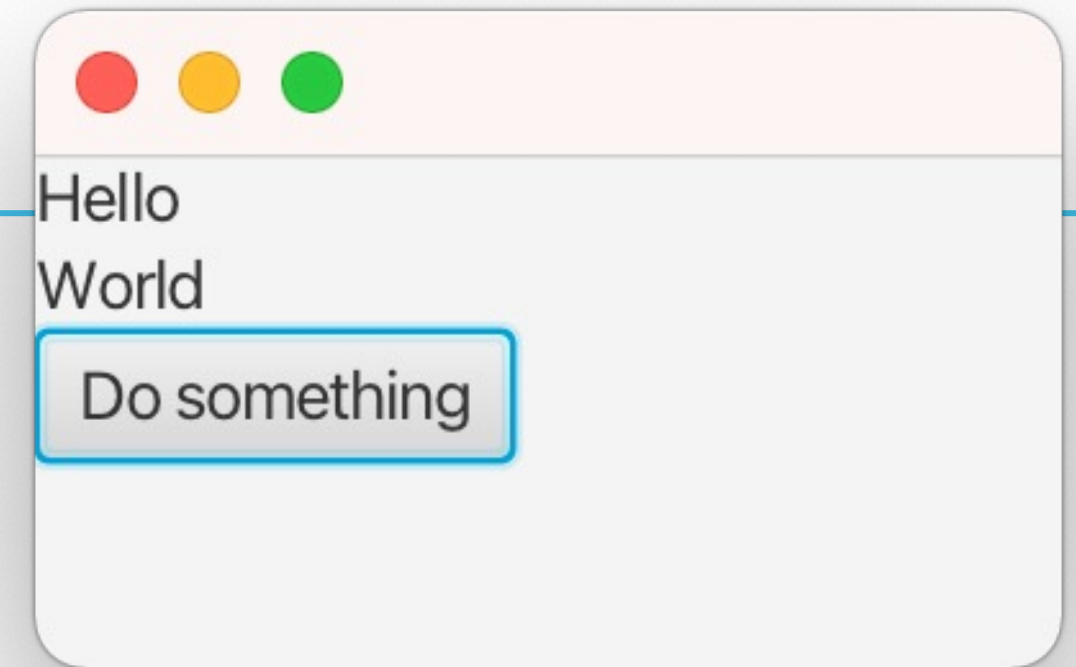
howest
hogeschool

The Controller

```
public class DemoController {
    @FXML private Label lblWord;

    public void doSomething(ActionEvent actionEvent) {
        lblWord.setText("user interaction");
    }
}
```

The controls in the the fxml-file need an fx:id , if you want to access them as a field in the controller.
Use the @FXML annotation, to make your fields private in the controller.

Some controls allow you to specify a handler method in the controller, using the #-symbol. Buttons, for instance, have the onAction-property.

Usually, you do not need to access the button itself, then you should not provide it with an fx:id . In case you do want to access the button, then you need the fx:id of course, but only add it if needed.

howest
hogeschool

Project structure

```java
@FXML private ListView<String> someLinesOfText;

public void doSomething(ActionEvent actionEvent) {
    lblWord.setText("user interaction");

    List<String> myClassicList = new ArrayList<>();
    myClassicList.add("a");
    myClassicList.add("b");
    myClassicList.add("c");

    someLinesOfText.setItems(FXCollections.observableList(
        myClassicList
    ));
```
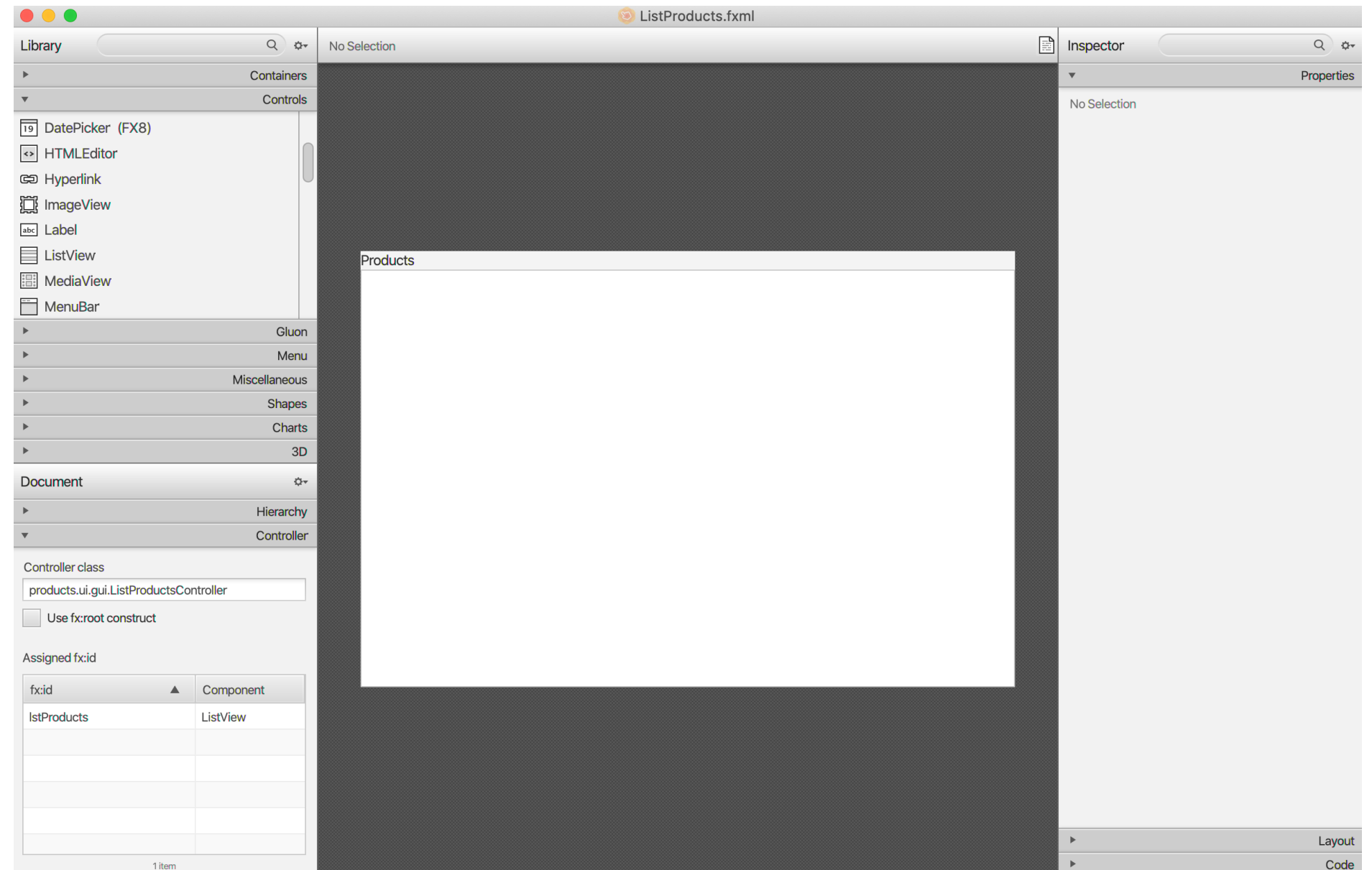
howest
hogeschool

SceneBuilder

Introducing SceneBuilder

- JavaFX application that can be used to design FX GUIs

- No longer required to write FXML manually

- Free download: https://gluonhq.com/products/scene-builder/

howest
hogeschool

Creating an FXML file with SceneBuilder

1. Create a new file (or open existing one)

2. Add a container (Containers)

3. Add controls (Controls)

4. Provide controls (and containers) with an fx:id (Code)

5. Provide controls with action handlers (Code)

6. Define the controller class (Controller)

howest
hogeschool
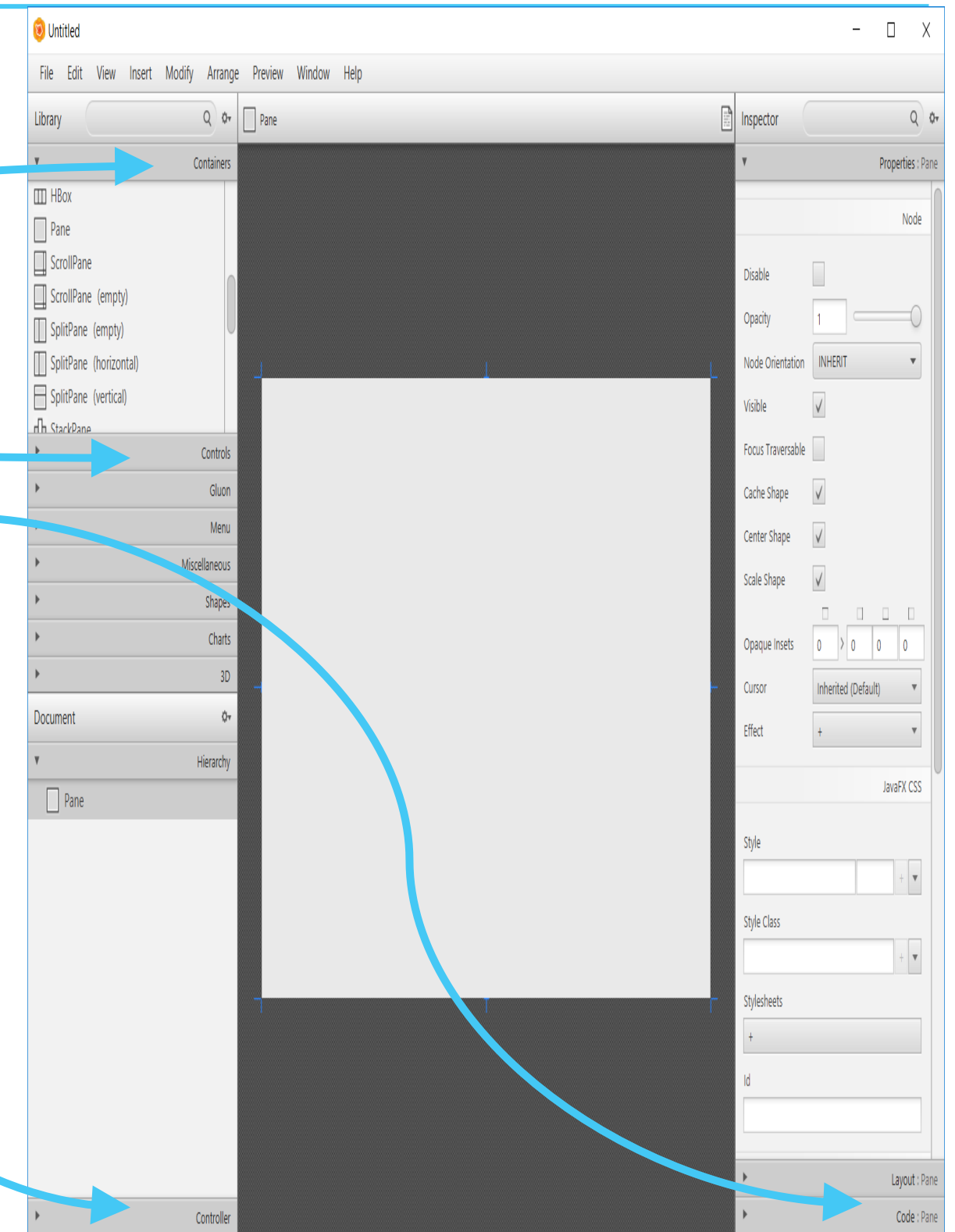
Creating an FXML file with SceneBuilder

1. Generate Controller class
   (View > Show Sample Controller Skeleton)

2. Copy-Paste in Java file
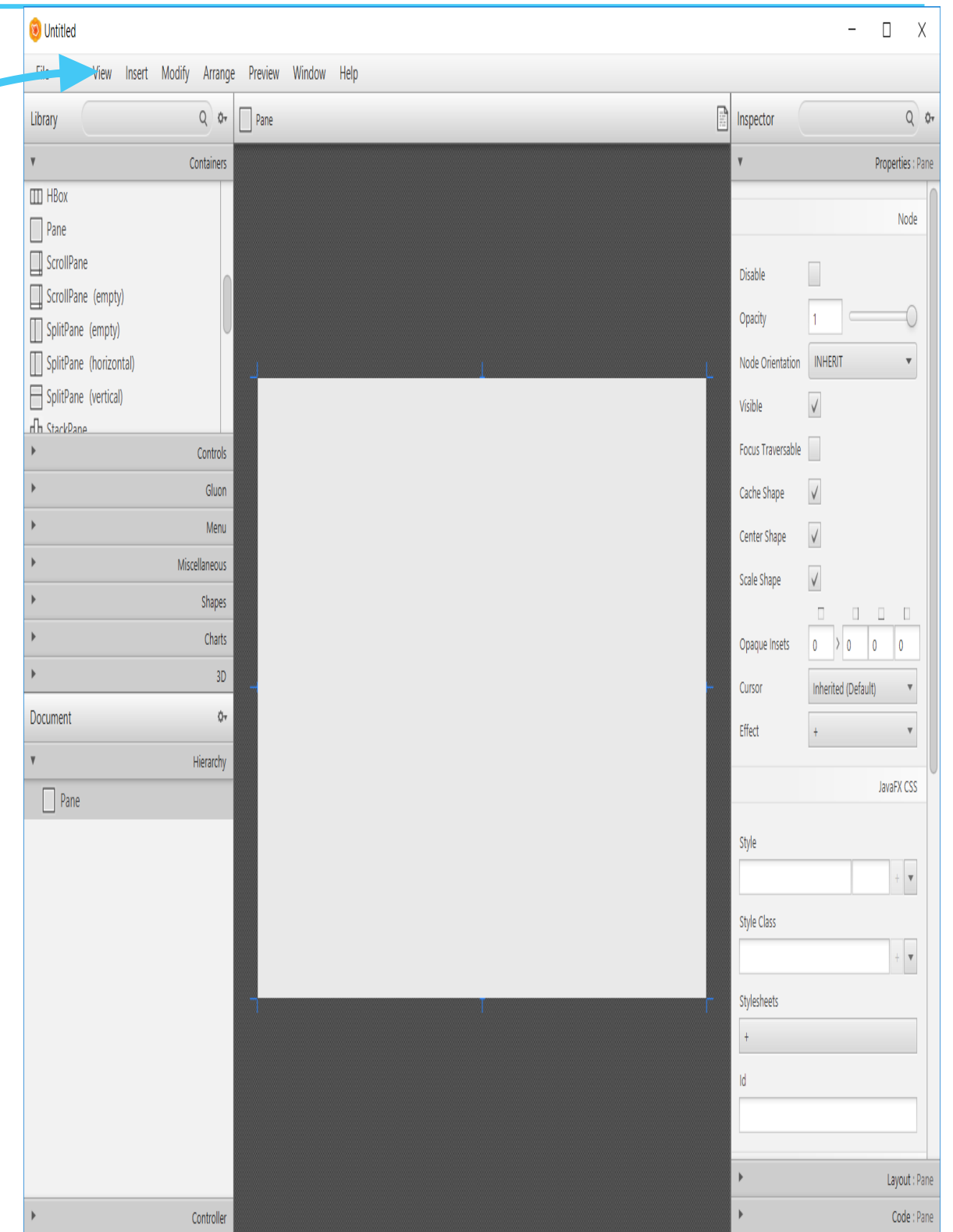
Alert dialog

Showing an Alert dialog

```java
@FXML
void doAdd(ActionEvent event) {
    try {
        String name = txtName.getText();
        double price = Double.parseDouble(txtPrice.getText());
        int vat = cboVAT.getSelectionModel().getSelectedItem();

        Product product = new Product(name, price, vat);

        Repositories.getProductsRepository().addProduct(product);
        products.add(product);
        stage.close();
    } catch (NumberFormatException ex) {
        Alert al = new Alert(Alert.AlertType.ERROR, "Invalid price specified.", ButtonType.CLOSE);
        al.showAndWait();
    } catch (ProductsException ex) {
        Alert al = new Alert(Alert.AlertType.ERROR, ex.getMessage(), ButtonType.CLOSE);
        al.showAndWait();
    }
}
```

howest
hogeschool
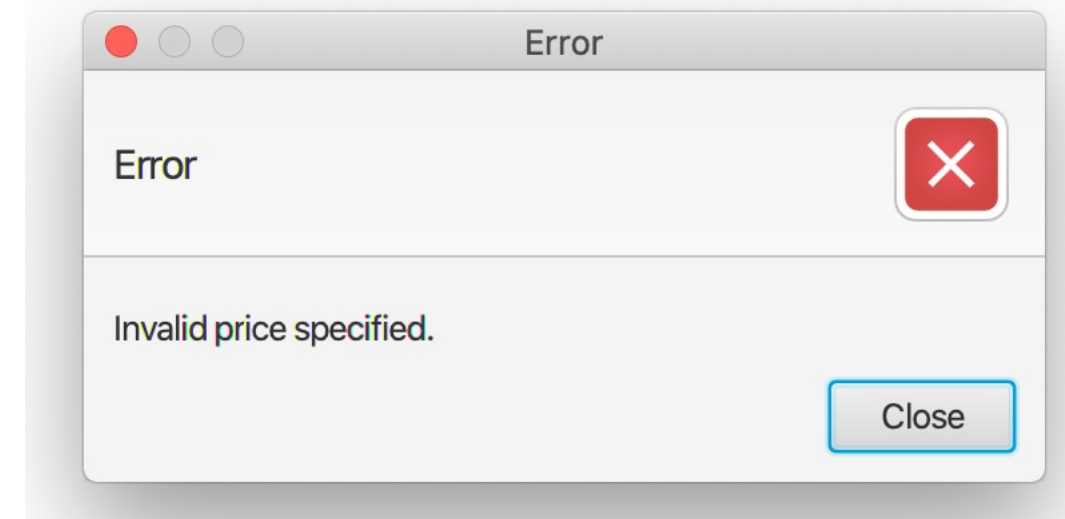
Common errors/mistakes

javafx.fxml.LoadException

- javafx.fxml.LoadException: Root hasn't been set. Use method setRoot() before load.

- Cause: sometimes SceneBuilder creates the following FXML:

    <fx:root ..... type="VBox" ...>

    </fx:root>

- Solution: replace by:

    <VBox ... >

    </VBox>

howest
hogeschool

java.lang.NullPointerException: Location is required.

- Wrong path/name of your fxml:

FXMLLoader.*load*(

      FXApp.**class**.getResource(

      "fxml/ShowProducts.fxml"

));

FXMLLoader.*load*(

      FXApp.**class**.getResource(

      "/fxml/ShowProdukts.fxml"

));

- FXML file not where it should be:

FXMLLoader.*load*(

      FXApp.**class**.getResource(

      "/fxml/ShowProducts.fxml"

));

Wrong or missing Controller definition in FXML file

- java.lang.ClassNotFoundException

  - Typo or non-existing controller class specified

- javafx.fxml.LoadException: No controller specified

  - No controller specified

- Initialize method is not executed

  - No controller specified

howest
hogeschool

java.lang.UnsupportedOperationException

```
List<Products> getAllProducts() {
        return Collections.unmodifiableList(allProducts);
}
```

FXCollections.observableList(getAllProducts());

FXCollections.observableList(new ArrayList<>(getAllProducts()));

FXCollections.observableArrayList(getAllProducts());

howest
hogeschool

FX:ids on Buttons

- For most buttons you will implement an "onAction" in the FXML and the corresponding method in the controller.
- If this is the use-case, you should not add an fx:id for this button, it is not needed!

howest
hogeschool