howest
hogeschool

Object Oriented Architectures and Secure Development

User interfaces

Arne Debou
Mattias De Wael
Frédéric Vlummens

howest
hogeschool

JavaFX!!!

What's JavaFX?

JavaFX is a Java-based software platform for creating rich and interactive desktop applications.

Features:

- UI Controls: A wide range of customizable UI elements.
- Scene Graph: Efficient rendering and flexible UI hierarchy.
- CSS Styling: Easy UI customization with CSS. (not for today)
- Media Support: Audio, video, and 2D/3D graphics. (not for today)
- Animation: Smooth transitions and effects. (not for today)
- FXML: Declarative UI design.
- Integration: Works with Swing. (not for today)
- Platform Independent: Runs on Windows, macOS, Linux. (always ;-))
- Open Source: Part of the OpenJFX project. (always ;-))

Use Cases: Business apps, games, multimedia, and more.

howest
hogeschool

JavaFX: Hello world

The Application class

```
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();
    }
}
```

Adding JavaFX dependencies to build.gradle

```
plugins {
    id 'java'
    id 'org.openjfx.javafxplugin' version ‘0.1.0'
}
```

```
javafx {
    version = “21"
    modules = [ 'javafx.controls']
}
```

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

howest hogeschool

Do not launch your app from IntelliJ

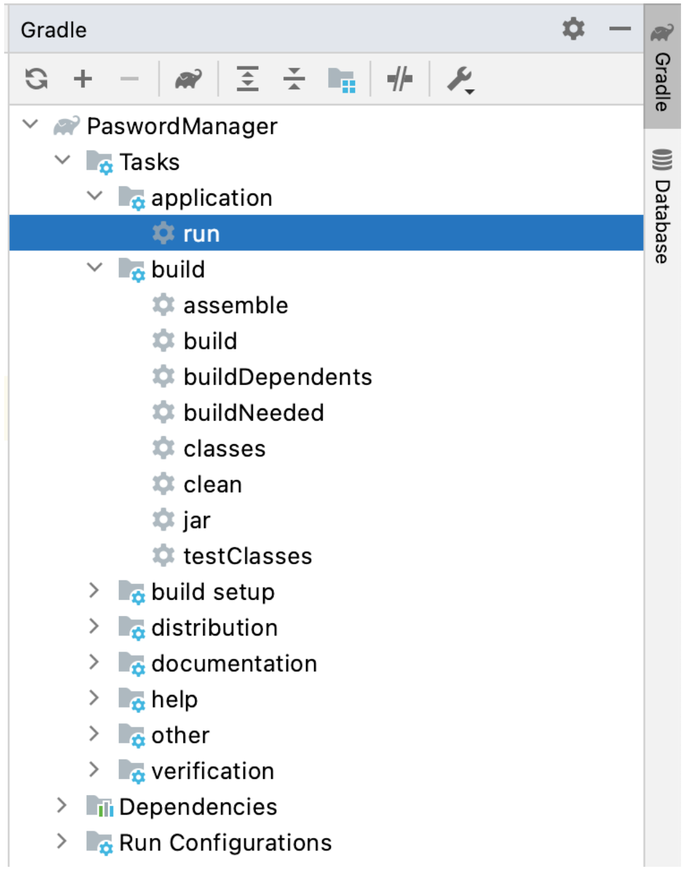
This will cause an error:

```
> 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
```

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)



howest hogeschool

howest
hogeschool

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**:

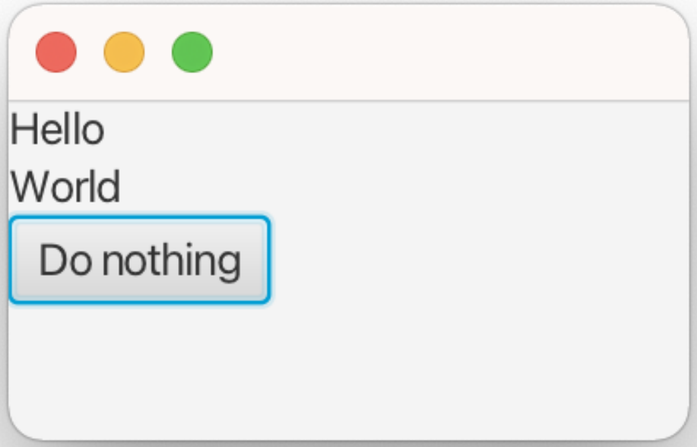
```
javafx {
    version = "18"
    modules = [ 'javafx.controls', 'javafx.fxml' ]
}
```


howest hogeschool

The FXML file

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

```
<?xml 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>
```

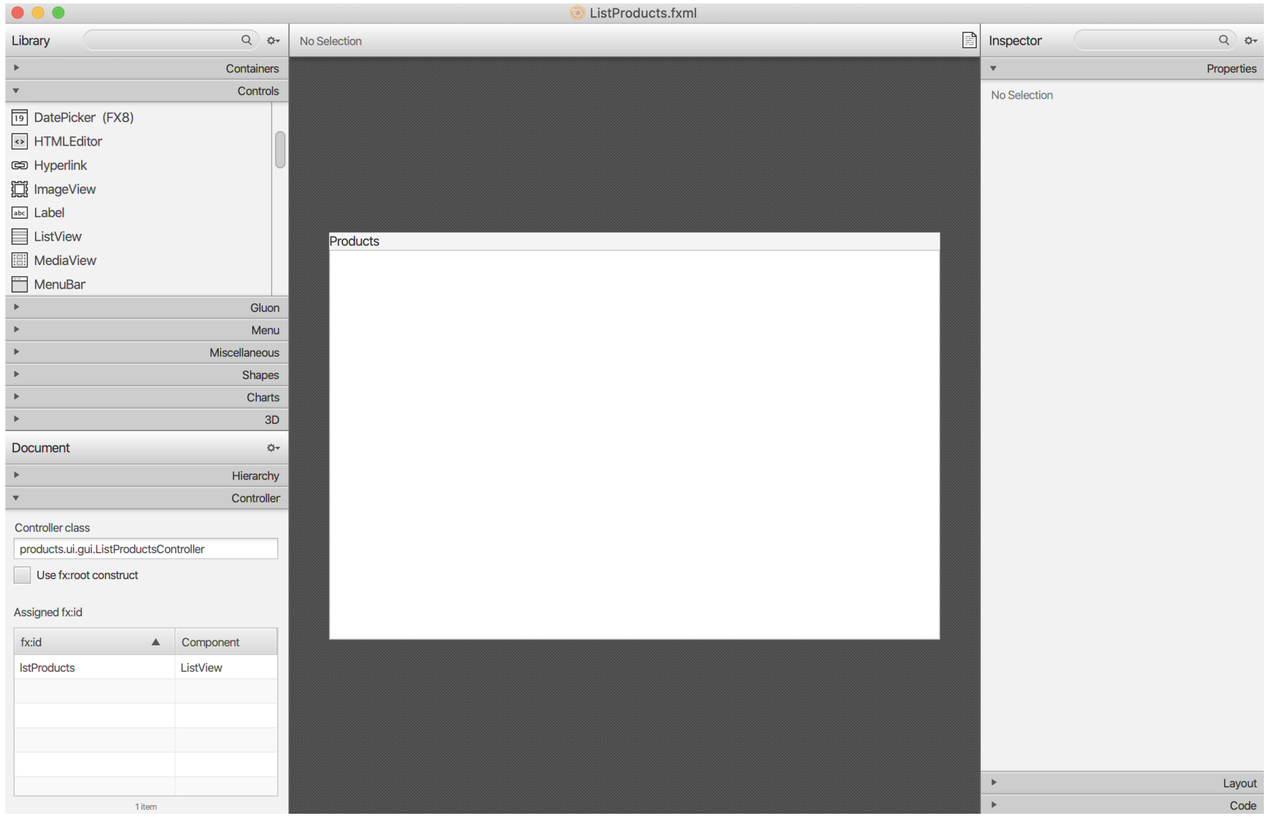


howest
hogeschool

SceneBuilder

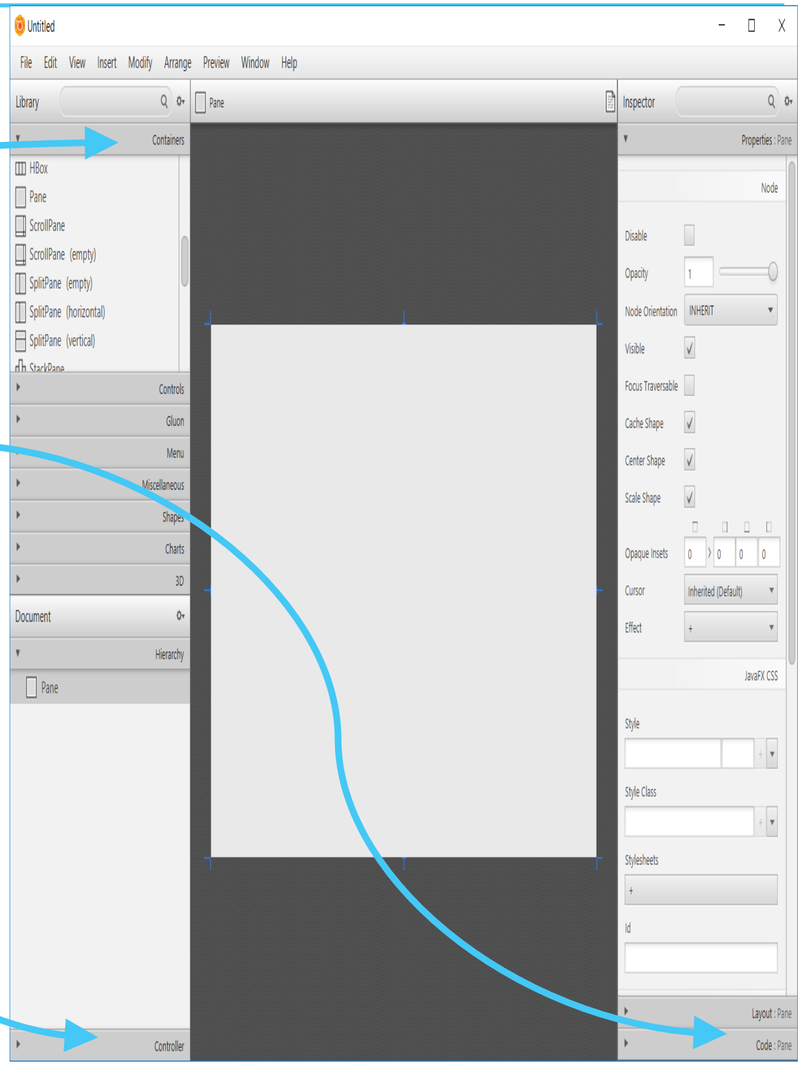
Introducing SceneBuilder

- 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/



Creating an FXML file with SceneBuilder

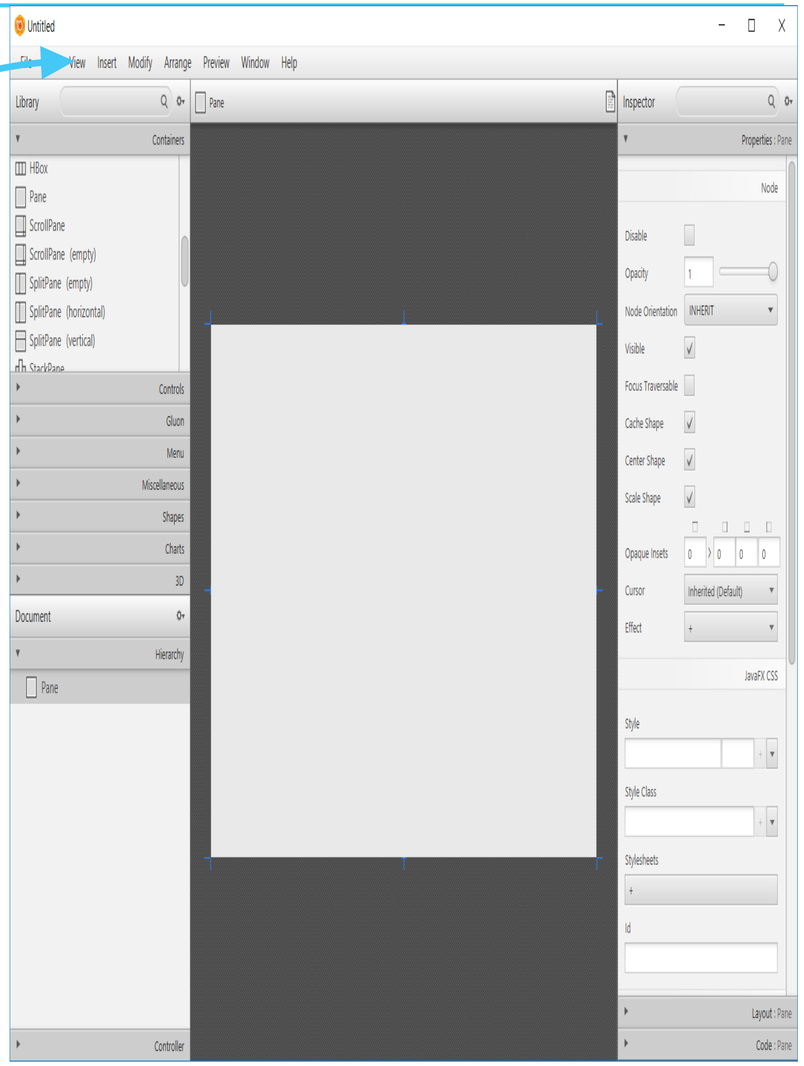
1. Create a new file (or open existing one)
2. Add a container (Containers)
3. Add controls (Controls)
4. Provide controls (and containers) with an fx:id (Code)
5. Provide controls with action handlers (Code)
6. Define the controller class (Controller)



howest hogeschool

Creating an FXML file with SceneBuilder

1. Generate Controller class
   (View > Show Sample Controller Skeleton)
2. Copy-Paste in Java file



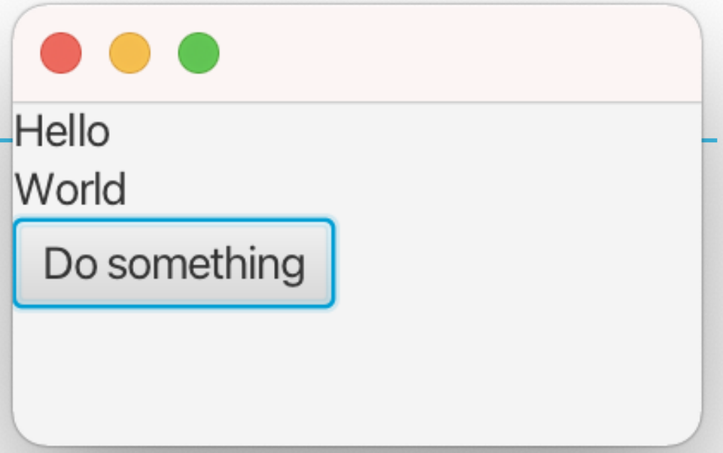
howest hogeschool

Load the FXML file in Application class

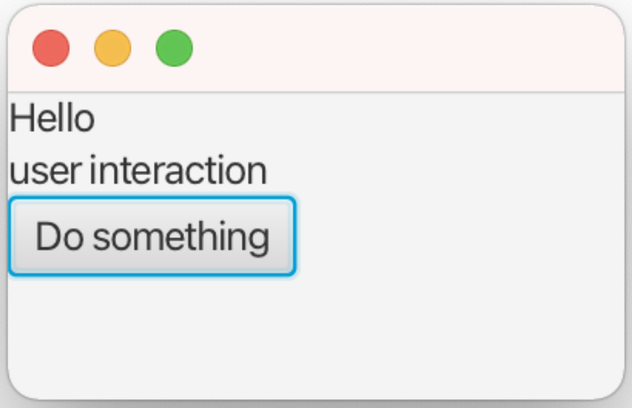
```
public class FxApplication extends Application {

    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 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.

Project structure

```
@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
    ));
```

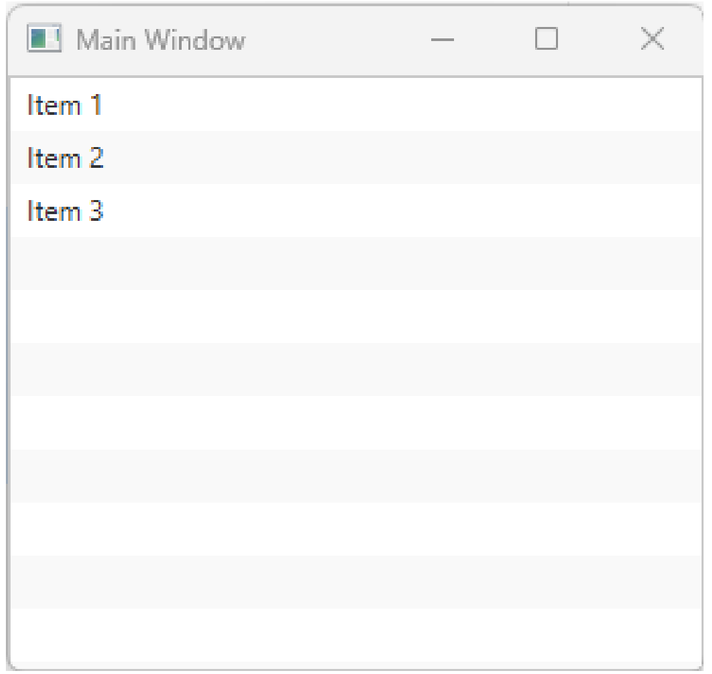

howest hogeschool

howest
hogeschool

Second screen

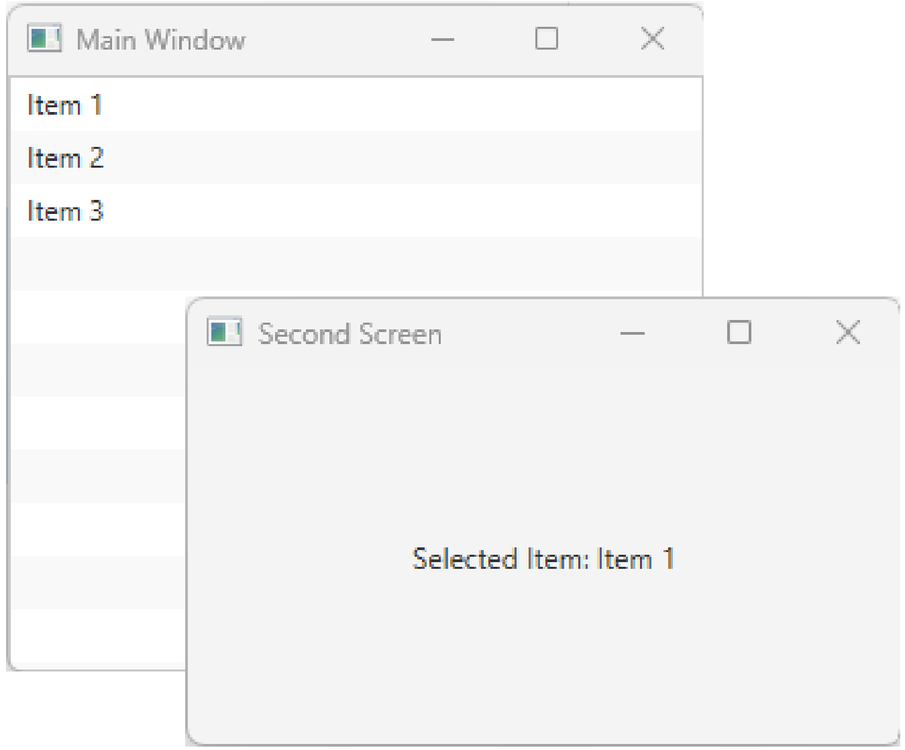
Second screen call up

```
listView.setOnMouseClicked(event -> {
    String selectedItem =
listView.getSelectionModel().getSelectedItem();
    if (selectedItem != null) {
        openSecondScreen(selectedItem);
    }
});
```



Second screen method

```
private void openSecondScreen(String selectedItem) {
    Stage secondStage = new Stage();
    secondStage.setTitle("Second Screen");

    // Create the content for the second screen
    StackPane secondScreenLayout = new StackPane();
    secondScreenLayout.getChildren().add(new Label("Selected Item: " + selectedItem));

    Scene secondScreenScene = new Scene(secondScreenLayout, 300, 150);
    secondStage.setScene(secondScreenScene);

    // Show the second screen
    secondStage.show();
}
```



Open a second window with contextual information

When you open a new window,

You often want to pass some information to that window.

Which is the same a saying: pass some information to the controller of the window.

One problem: we cannot access the controller using the “old” technique.

```
@Override
  public void start(Stage primaryStage) throws Exception {
      Parent parent = FXMLLoader.load(Program.class.getResource("/fxml/ChatListView.fxml"));
      Scene scene = new Scene(parent);
      primaryStage.setScene(scene);
      primaryStage.show();
  }
```


howest hogeschool

Open a second window with contextual information

When you open a new window,

You often want to pass some information to that window.

Which is the same a saying: pass some information to the controller of the window.

One problem: we cannot access the controller using the “old” technique.

We study two techniques:

- ask the FXMLLoader to give you the controller it created.
- create a controller yourself and ask the FXMLLoader to use that one.

howest hogeschool

Open a second window with contextual information

We cannot access the controller using the “old” technique.

We study two techniques:

- ask the FXMLLoader to give you the controller it created.
- create a controller yourself and ask the FXMLLoader to use that one.

Both techniques require you to create the FXMLLoader upfront:

```
FXMLLoader loader = new FXMLLoader(
        getClass().getResource("/path/to/some/file.fxml")
);
```


howest hogeschool

Ask the FXMLLoader to give you the controller it created.

```
FXMLLoader loader = new FXMLLoader(
        getClass().getResource("/path/to/some/file.fxml")
);

Parent parent = loader.load(); // load FXML first
SomeDedicatedController controller = loader.getController(); // then retrieve controller

controller.doWhateverIsNeeded(extraData); // pass data …
```


howest hogeschool

Create a controller and ask the FXMLLoader to use that one

```
FXMLLoader loader = new FXMLLoader(
        getClass().getResource("/path/to/some/file.fxml")
);


// first create ...
SomeDedicatedController controller = new SomeDedicatedController(initData);
loader.setController(controller); // ... and set the controller

Parent parent = loader.load(); // then load the FXML

controller.doWhateverIsNeeded(extraData); // pass extra data
```


howest hogeschool

Create a controller and ask the FXMLLoader to use that one

```
FXMLLoader loader = new FXMLLoader(
        getClass().getResource("/path/to/some/file.fxml")
);
loader.setController(controller); // set the controller
Parent parent = loader.load(); // then load the FXML
```

In this case there is no need (not allowed) to specify a controller class in the FXML.

howest hogeschool

A window is represented as a stage.

We can open and close stages:

```
stage.show();
stage.showAndWait(); // until it is closed.
stage.close();
```

We can access the stage by ‘looking it up’ or pass it along from elsewhere:

```
Stage currentStage = (Stage) anyElement.getScene().getWindow();
```

```
// in a controller:
public void setStage(Stage stage) {
    this.stage = stage;
}
```


howest hogeschool

You do not need to create a new window:

Until now, we always added the FXML-view into a new scene and into a (new) stage.

```
@Override
  public void start(Stage primaryStage) throws Exception {
     Parent parent = FXMLLoader.load(Program.class.getResource("/fxml/ChatListView.fxml"));
     Scene scene = new Scene(parent);
     primaryStage.setScene(scene);
     primaryStage.show();
  }
```

But we can also add the FXML-view to an existing container:

```
FXMLLoader loader = new FXMLLoader(
      ChatListController.class.getResource("/fxml/ChatListEntryView.fxml")
);

Parent node = loader.load();
someVBox.getChildren().add(node);
```


howest hogeschool

howest
hogeschool

i18n

What's i18n

- "Internationalization" in Java
- Support of different languages and regions.

- Locale-class, subtract local sources from the application, f.e.:
  - Date & time notations
  - Language
  - Currency

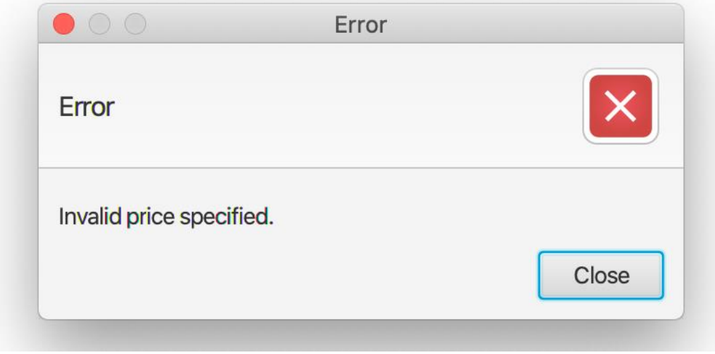
- ResourceBundles

howest
hogeschool

Alert dialog

Showing an Alert dialog

```
@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

java.lang.UnsupportedOperationException

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

```
FXCollections.observableList(getAllProducts());
```

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

```
FXCollections.observableArrayList(getAllProducts());
```

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!