Java GUI: JavaFX, FXML, CSS

Tamás Ambrus, István Gansperger

Eötvös Loránd University
ambrus.thomas@gmail.com

Introduction

GUI libraries

Java has many GUI frameworks such as AWT, Swing or SWT.
These are old solutions, but still used in the industry.
For example:

- SWT: if you are developing an Eclipse plugin, you can only use SWT,
- Swing: newer than AWT, good choice for any not embedded Java applications.

GUI libraries

Java has many GUI frameworks such as AWT, Swing or SWT.
These are old solutions, but still used in the industry.
For example:

- SWT: if you are developing an Eclipse plugin, you can only use SWT,
- Swing: newer than AWT, good choice for any not embedded Java applications.

It also has to be said that Swing is not the best choice for MVC applications.

MVC: Model-View-Controller

MVC is a software-architectural pattern for designing GUI applications. It divides the production code into 3 pieces:

- Model: the data of our app. This tier has the business logic through which controller updates the view.
- View: the tier of GUI components. GUI related code should be here mostly (view description files).
- Controller: this is the bridge between the two tiers. Converts commands for the target side.

To make our application fit to MVC, **we will use only 3 packages in GUI apps** from now on: view, controller, model.

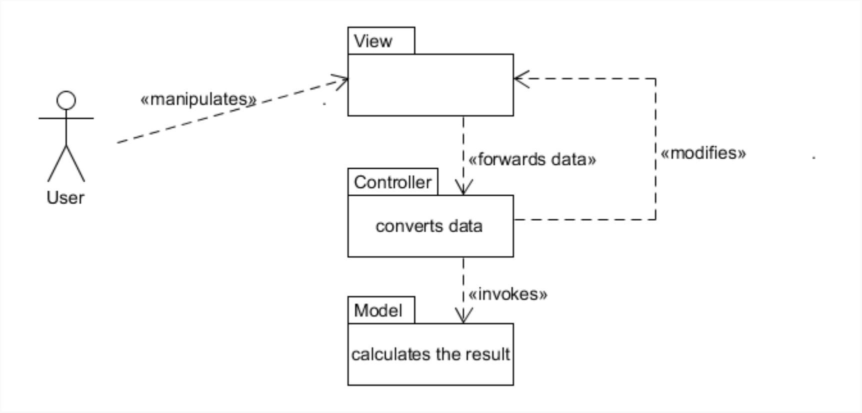
"Must not"s

Things that you must not do using MVC architecture:

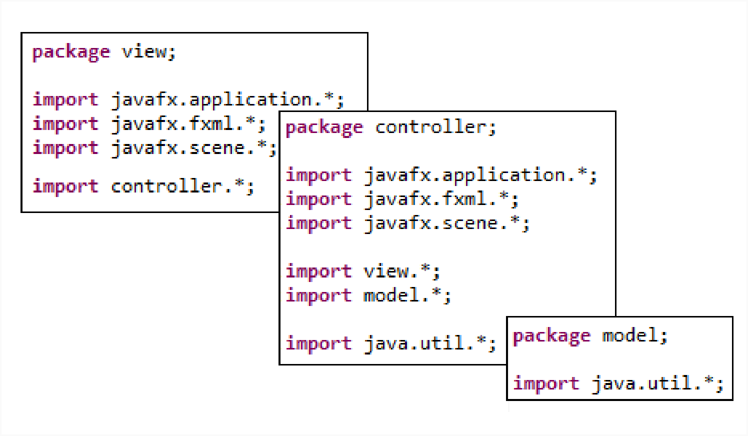
- store GUI related code outside the view or controller packages,
- store business logic code outside the model package,
- store data type classes, operations outside the model package,
- reach model from view,
- reach controller or view from model.

What you can do using MVC

Users can access only the view part, view part can access only controller part, controller part can access only model part of the app and modify the view, nothing else.



What you can do using MVC



```
package view;

import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;

import controller.*;
```

```
package controller;

import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;

import view.*;
import model.*;

import java.util.*;
```

```
package model;

import java.util.*;
```

Advantages of MVC

MVC is an architecture design you should follow, since:

- separates code well in a complex application (you can find easily what you search),
- most code of tiers can be replaced separately: if you want to change the model implementation, you can do it without touching your view,
- enables the developers to work on individual components.

Swing → JavaFX

Swing is currently in a process of being replaced by another library named JavaFX.
This way JavaFX is intended to be used as the standard GUI library.

Let's learn JavaFX basics!

JavaFX

About

JavaFX is a good choice for

- desktop applications,
- web applications,
- Android, iOS, Raspberry PI.

About

JavaFX is a good choice for

- desktop applications,
- web applications,
- Android, iOS, Raspberry PI.

Key features are:

- Java APIs: in this library classes and interfaces are written in Java code so JavaFX is designed to be a friendly alternative to JVM languages, e.g. Scala.
- FXML: **FXML is an XML-based** declarative markup language for constructing a JavaFX app user interface.

About

What makes our development easier and nicer:

- Built-in UI controls and **CSS**: JavaFX provides all major components to develop GUI applications. They can be skinned with CSS files.
- The xml-css-java triplet separates code very good.
- **SceneBuilder** is a WYSIWYG application that generates FXML markup that can be ported to an IDE.

FXML

FXML is an XML-based language that provides the structure for building a user interface separate from the application logic of your code.

- FXML maps directly to Java
  - in Eclipse you can use Ctrl+Space to look for allowed elements and attributes
- you can set the appropriate css and controller files in FXML files

Basic components

The basic components we will work with are the following:
Label, Button, RadioButton, CheckBox, TextField, ComboBox.

So, do we just need to drop them with correct positions and sizes?

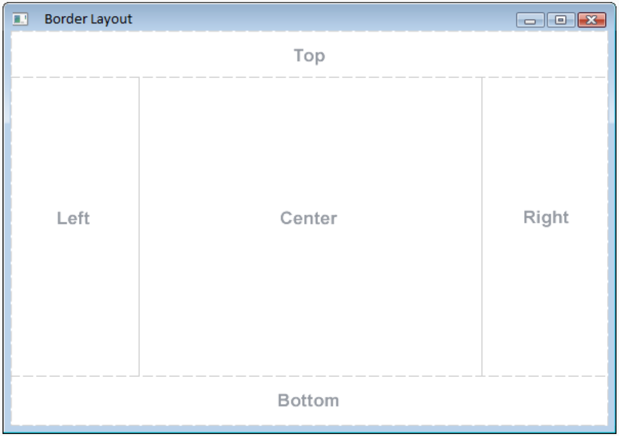
HBox and VBox

HBox: a layout pane that supports the arrange of components in a single row.

VBox: supports the arrange of components in a single column.

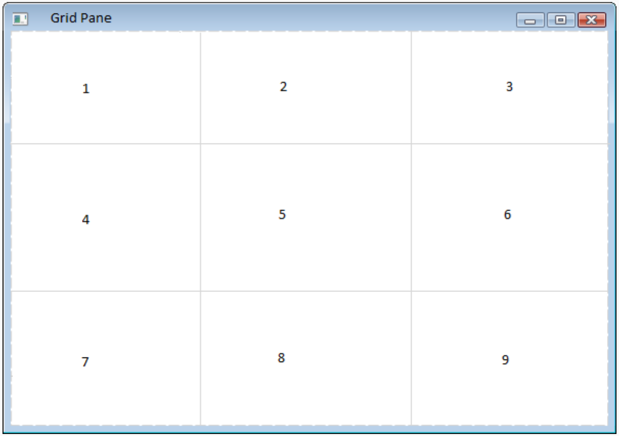
BorderPane

BorderPane provides five regions to place components into: top, bottom, left, right, center. These regions can be any size.



GridPane

GridPane enables to create flexible grid of rows and columns. It is useful to create tables or any forms that should be arranged into a grid.



FXML example with controller

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

<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>

<BorderPane xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="controller.Controller">
  <top>
    <TextField text="textfield:>" />
  </top>
  <center>
    <GridPane fx:id="numbersPane">
      <children>
        <Button text="1" GridPane.columnIndex="0"
           GridPane.rowIndex="0" />
        <Button text="2" GridPane.columnIndex="0"
           GridPane.rowIndex="1" />
      </children>
    </GridPane>
  </center>
</BorderPane>
```

Controller, id

In the previous example we specified the controller and the id of the stored GridPane. This enables the manipulation of that element in the controller class. We need only to store as a class field with the @FXML annotation.

```
package controller;

public class Controller {

    @FXML
    private GridPane numbersPane;

}
```

Work with components

This way we can manipulate components in methods of controller, e.g.

```
package controller;

public class Controller implements Initializable {

    @FXML
    private GridPane numbersPane;

    @Override
    public void initialize(URL location, ResourceBundle res) {
        numbersPane.add(new Button("3"), 1, 0);
    }
}
```

Initializable interface

Controllers may implement the Initializable interface whose only responsibility is to provide an initialize method in which we can init our components.

- GUI components are initialized here instead of in a constructor
- this method is called to initialize a controller after its root element has been completely processed

JavaFX CSS

JavaFX CSS

JavaFX Cascading Style Sheets (CSS) is based on the W3C CSS version 2.1 with some additions from current work on version 3. The syntax can be familiar to web developers, there are a few difference though:

- attributes start with "-fx-"
- some attribute names are different

```
.button {
    -fx-pref-width: 50.0px;
    -fx-border-style: dotted;
}
```

JavaFX CSS cheat sheet

First of all, you can solve your CSS problems with these:

- pressing *Ctrl+Space* anywhere in the CSS file
- visiting this site: `https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html`
- google on it; stackoverflow will most likely show an answer to your problem

JavaFX CSS

Some components have default style class names:

- Button: *.button* { *-fx-...* },
- TextFields: *.text-field* { *-fx-...* },
- RadioButton: *.radio-button* { *-fx-...* }.

```
<Button text="I will be styled" />
```

JavaFX CSS

Some components don't have default style class names; for those, you have to define a style class:

- GridPane: "*Style class: empty by default*" - CSS Ref. Guide

CSS:

```css
.redBackgrounded {
    -fx-background-color: red;
}
```

FXML:

```xml
<GridPane styleClass="redBackgrounded" />
<Label text="no styled label" />
<Button styleClass="redBackgrounded"
  text="I'm red, even though I have an own style class" />
```

In general, anything can have a customized style class.

JavaFX CSS

There are more functionalities regarding the CSS files:

- if you need attributes to be common for more style classes: *.sc1, .sc2* { *-fx-...* }
- hierarchy, if you need to affect only *sc2* components that are placed in *sc1*: *.sc1 .sc2* { *-fx-...* }
- pseudo-classes, states of components of the given style class:

```
.button:hover {
    // what style should buttons have if the mouse is over them
}
```

References

- https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/Initializable.html
- https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
- http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm
- http://docs.oracle.com/javafx/2/layout/jfxpub-layout.htm