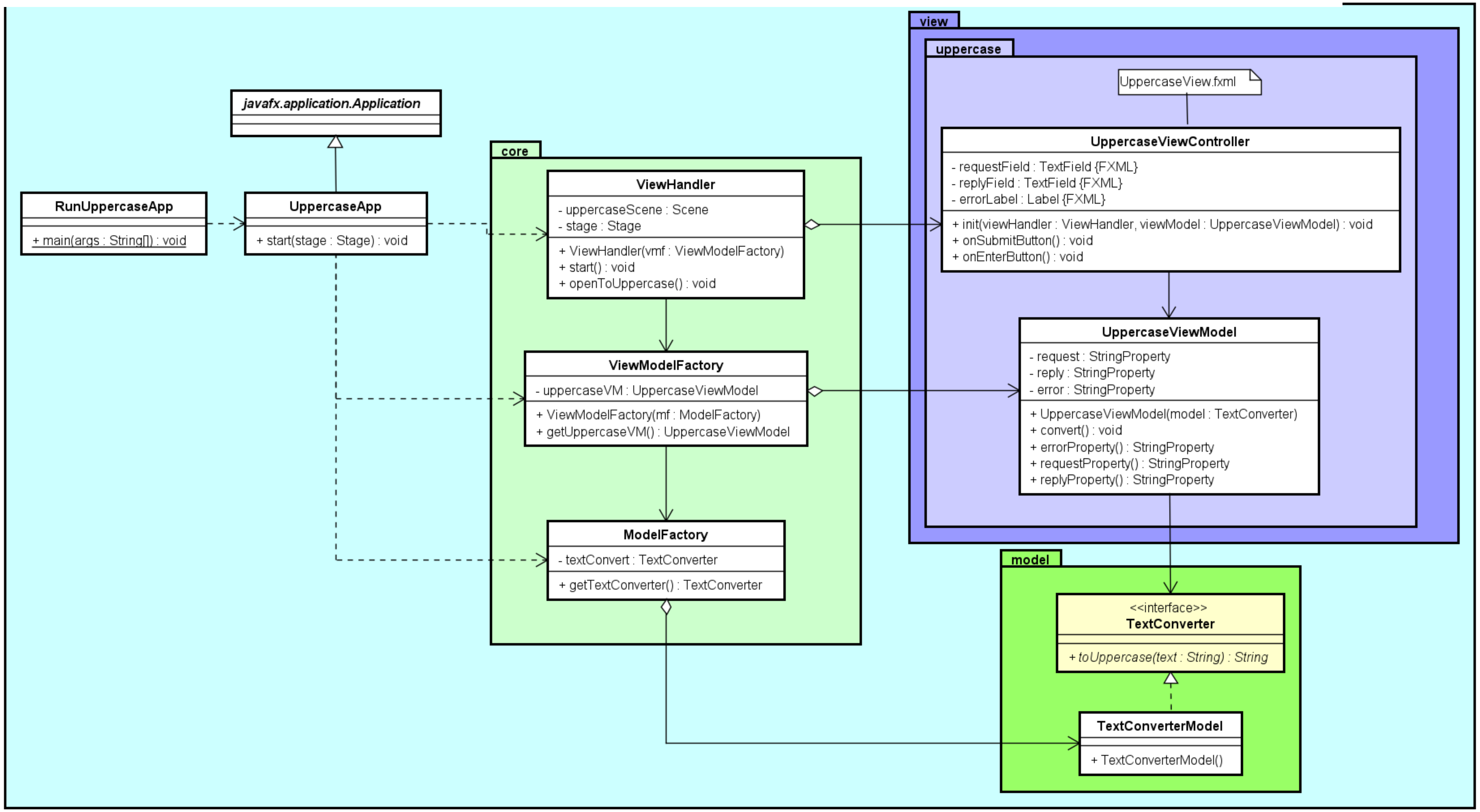
Upper case MVVM

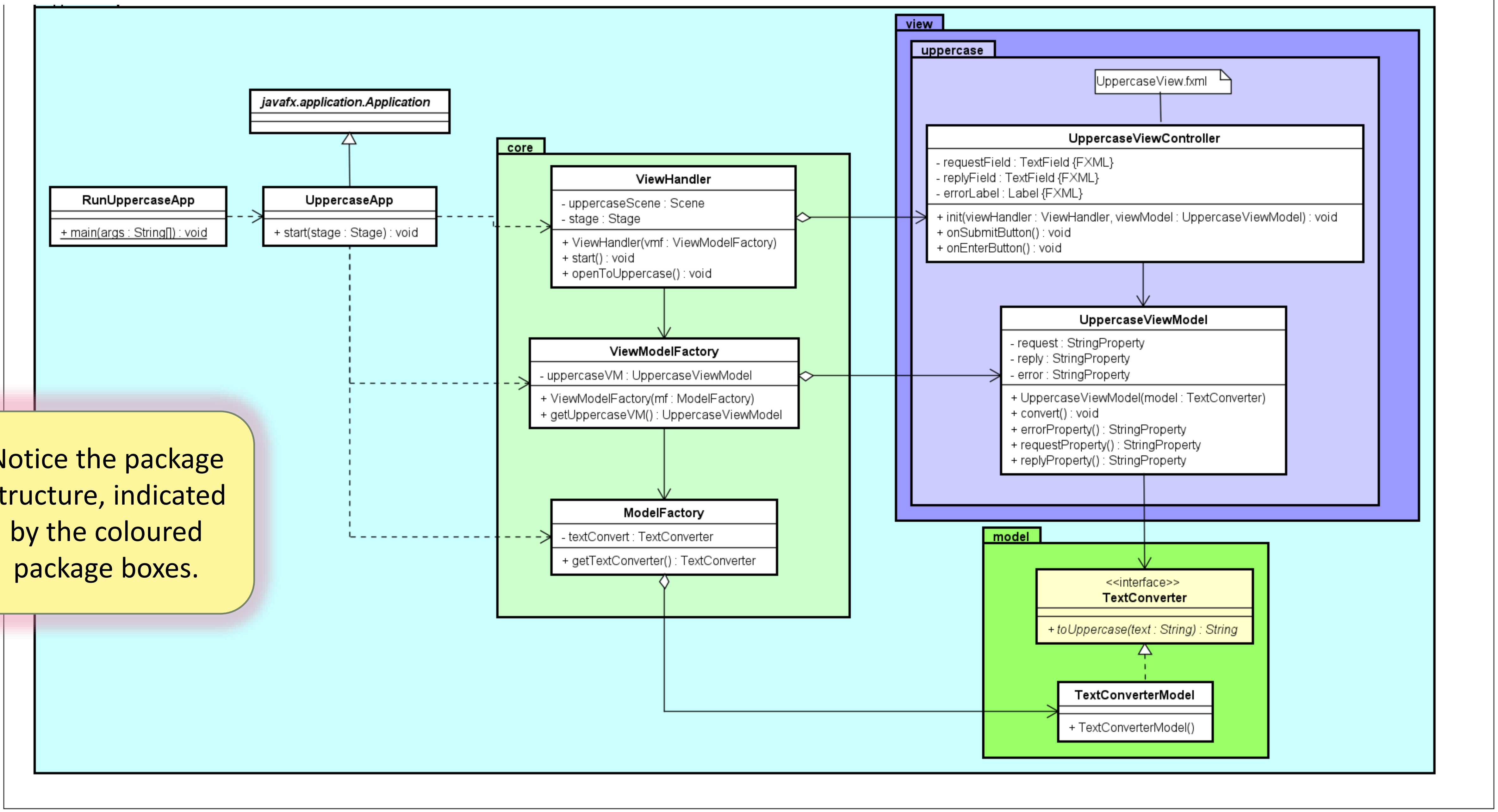
This tutorial will have you create a super simple MVVM application, which can convert a String input to upper case. The main purpose is to practice the package and class structure.

This is the class diagram of the end result



javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

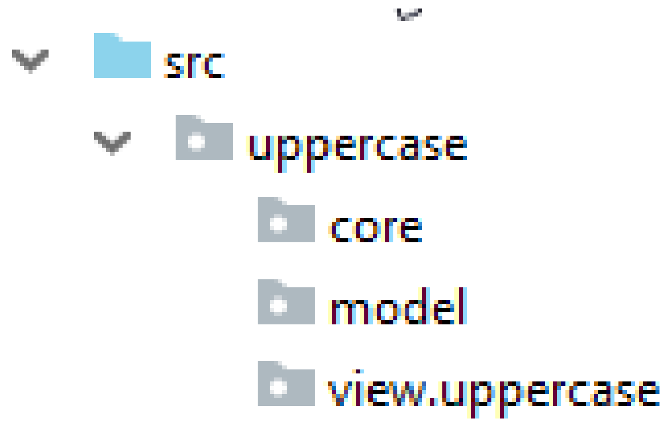
This is the class diagram of the end result

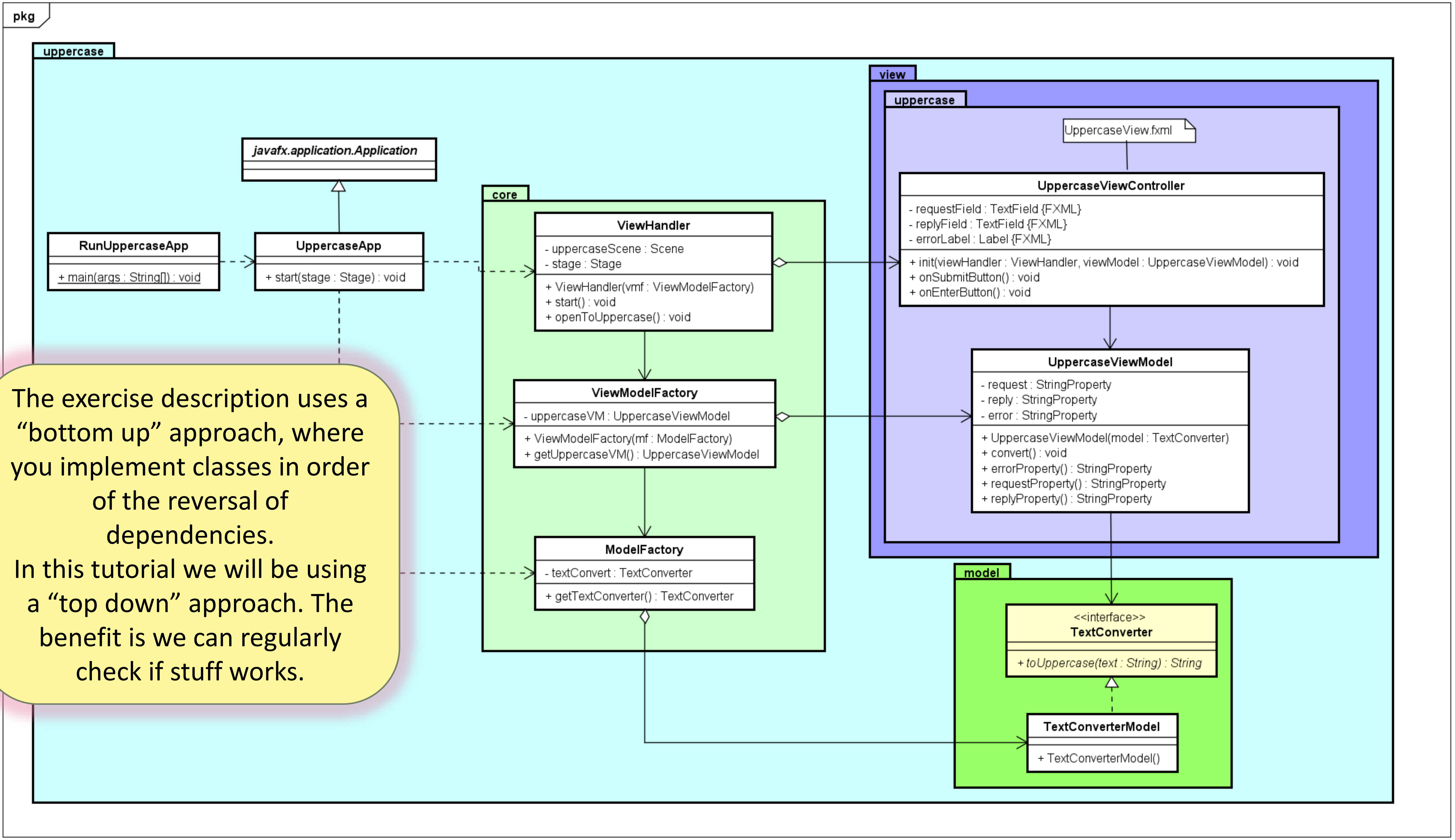

javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

Notice the package structure, indicated by the coloured package boxes.

New project

- Start by creating a new project, or module, or just a package for this program.
- Then you create the following package structure, see figure to the right.
- The core will always contain the four couple of classes in every mvvm project
- The model may contain sub-packages, if you in larger projects have more models
- The view package will contain a sub-package for each view or feature.

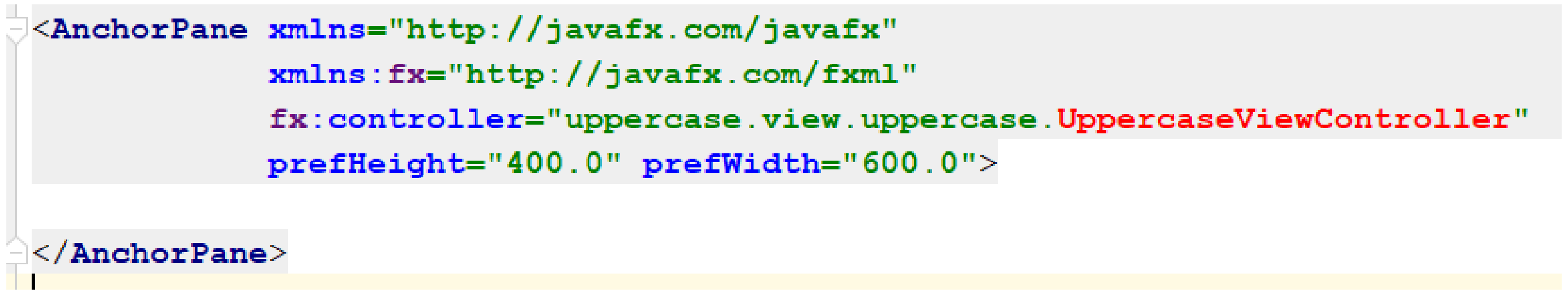



pkg
uppercase
javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

The exercise description uses a "bottom up" approach, where you implement classes in order of the reversal of dependencies.
In this tutorial we will be using a "top down" approach. The benefit is we can regularly check if stuff works.

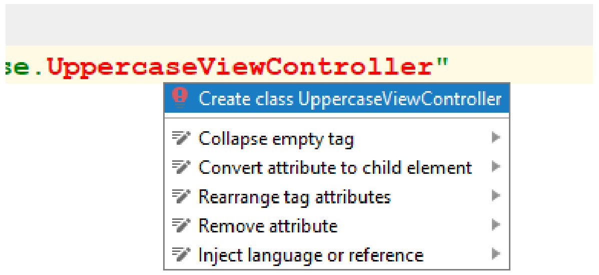
- Start by creating a new fxml file inside the uppercase.view.uppercase package, call it UppercaseView.fxml
- It will probably open the file and highlight the fx:controller field.
- Rename this to uppercase.view.uppercase.UppercaseView**Controller**

- It's currently red, because the class doesn't exist:



```
<AnchorPane xmlns="http://javafx.com/javafx"
            xmlns:fx="http://javafx.com/fxml"
            fx:controller="uppercase.view.uppercase.UppercaseViewController"
            prefHeight="400.0" prefWidth="600.0">

</AnchorPane>
```

- Place the caret on the red UppercaseViewController and press alt + enter

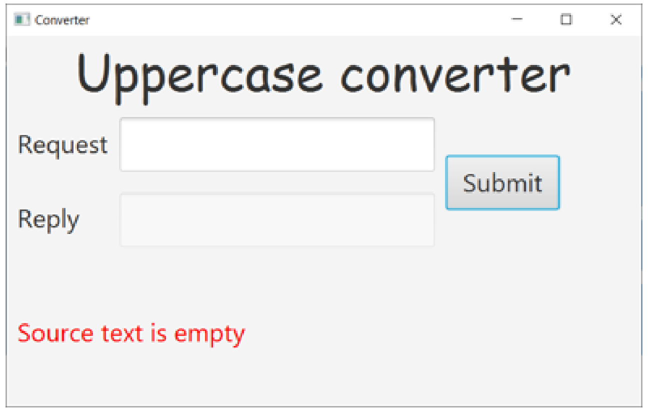


```
se.UppercaseViewController"
```

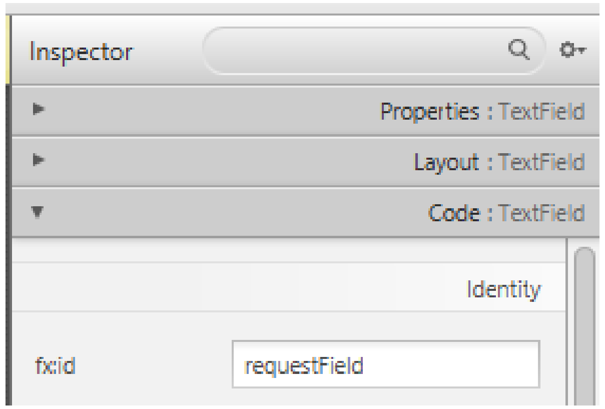
Create class UppercaseViewController
Collapse empty tag
Convert attribute to child element
Rearrange tag attributes
Remove attribute
Inject language or reference

- Select the Create class.

- Open the UppercaseView.fxml in Scenebuilder, and design something like this (exact design is not necessary, but the same elements must be present):

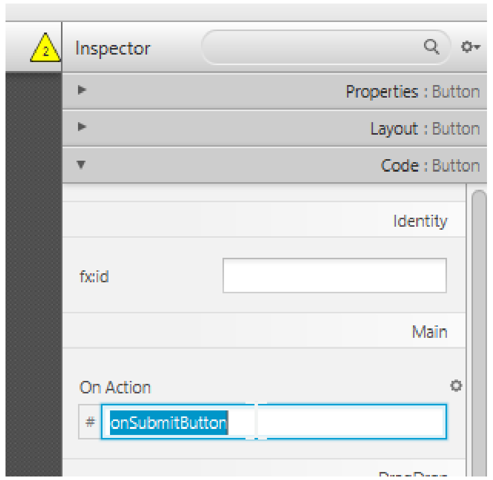


Source text is empty



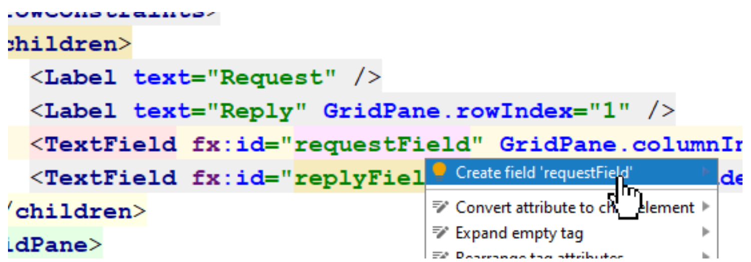
fx:id requestField

- Figure out the right containers.
- Use Labels, TextFields and a button.
- On the TextFields, and error label, insert the fx:id
- And on the button, put an action



On Action # onSubmitButton

- Back in IntelliJ, open the UppercaseView.fxml file.
- Find the fx:id="requestField", it should be highlighted in yellow to indicate this field variable is not found in the *controller*. Click the text, press alt+enter:



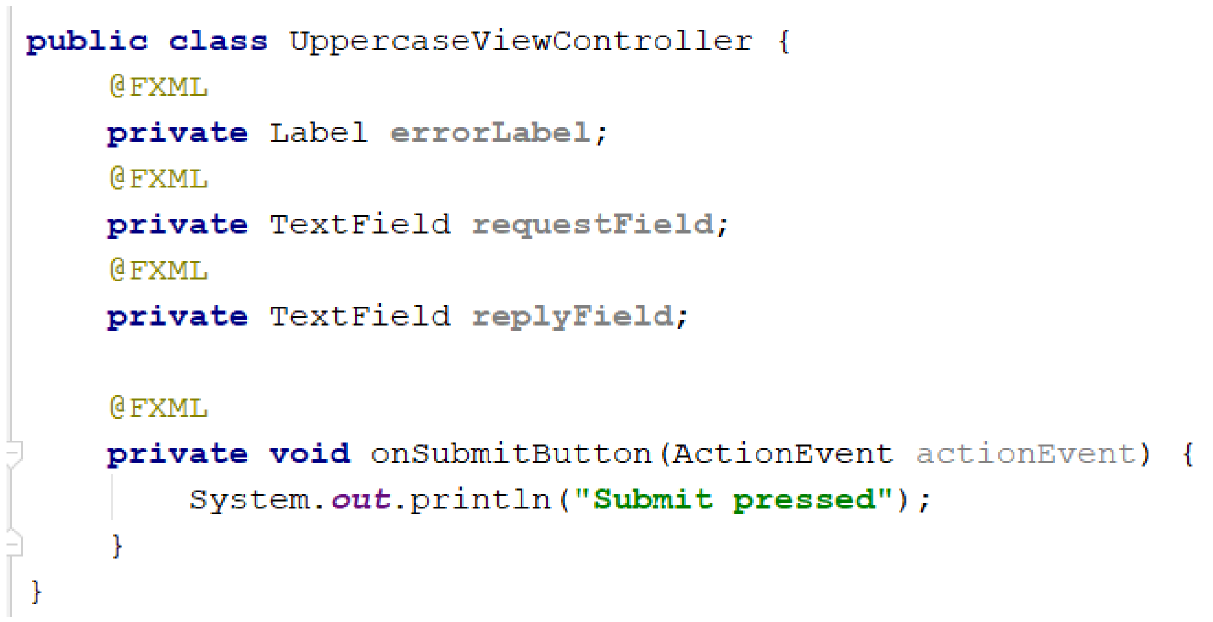
```
hildren>
  <Label text="Request" />
  <Label text="Reply" GridPane.rowIndex="1" />
  <TextField fx:id="requestField" GridPane.columnIn
  <TextField fx:id="replyFiel
/children>
dPane>
```

- Click the Create field, which should auto generate this in your UppercaseViewController.
- Do this for requestField, replyField, errorLabel, and onSubmitButton.
- By default, the resulting methods and fields are public. Change them all to private, and mark them with @FXML. See next slide for result

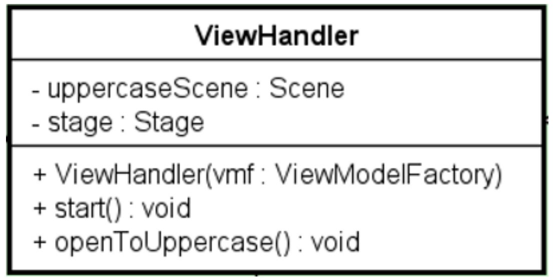
Your UppercaseViewController

- Insert a printout in the onSubmitButton method.

- The next step is to create the core, so we can run the program. The idea is that for each step, we can verify the code still behaves as expected.



```
public class UppercaseViewController {
    @FXML
    private Label errorLabel;
    @FXML
    private TextField requestField;
    @FXML
    private TextField replyField;

    @FXML
    private void onSubmitButton(ActionEvent actionEvent) {
        System.out.println("Submit pressed");
    }
}
```



- In the core package, create the ViewHandler class.
- Ignore the *constructor* for now.
- Create the fields shown in the UML, initialize the stage in the start() method.
- Create the *openToUppercase()* method, see next slide.

```
public void openToUppercase() {
    if(uppercaseScene == null) {
        try {
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(getClass().getResource("../view/uppercase/UppercaseView.fxml"));
            Parent root = loader.load();
            stage.setTitle("Upper case");
            uppercaseScene = new Scene(root);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    stage.setScene(uppercaseScene);
    stage.show();
}
```

This check is to see, if the Scene has been loaded before, in which case we reuse it. Lazy instantiation.

The path is relative to the location of ViewHandler. First we go up one folder, then into the view folder, then uppercase folder, and here we find the .fxml file

Setting title of window

Instantiating the Scene with the content of the .fxml file.

Inserting scene into the stage.

- In the *start()* method call the *openToUppercase()* method.

```
public void start() {
    stage = new Stage();
    openToUppercase();
}
```

- Create a class in *uppercase* package (the root package) called UppercaseApp.
- It should extend javafx.application.Application
- Override the *start()* method
- Inside the start() method:
  - Instantiate a ViewHandler
  - Call *start()* on the viewHandler.

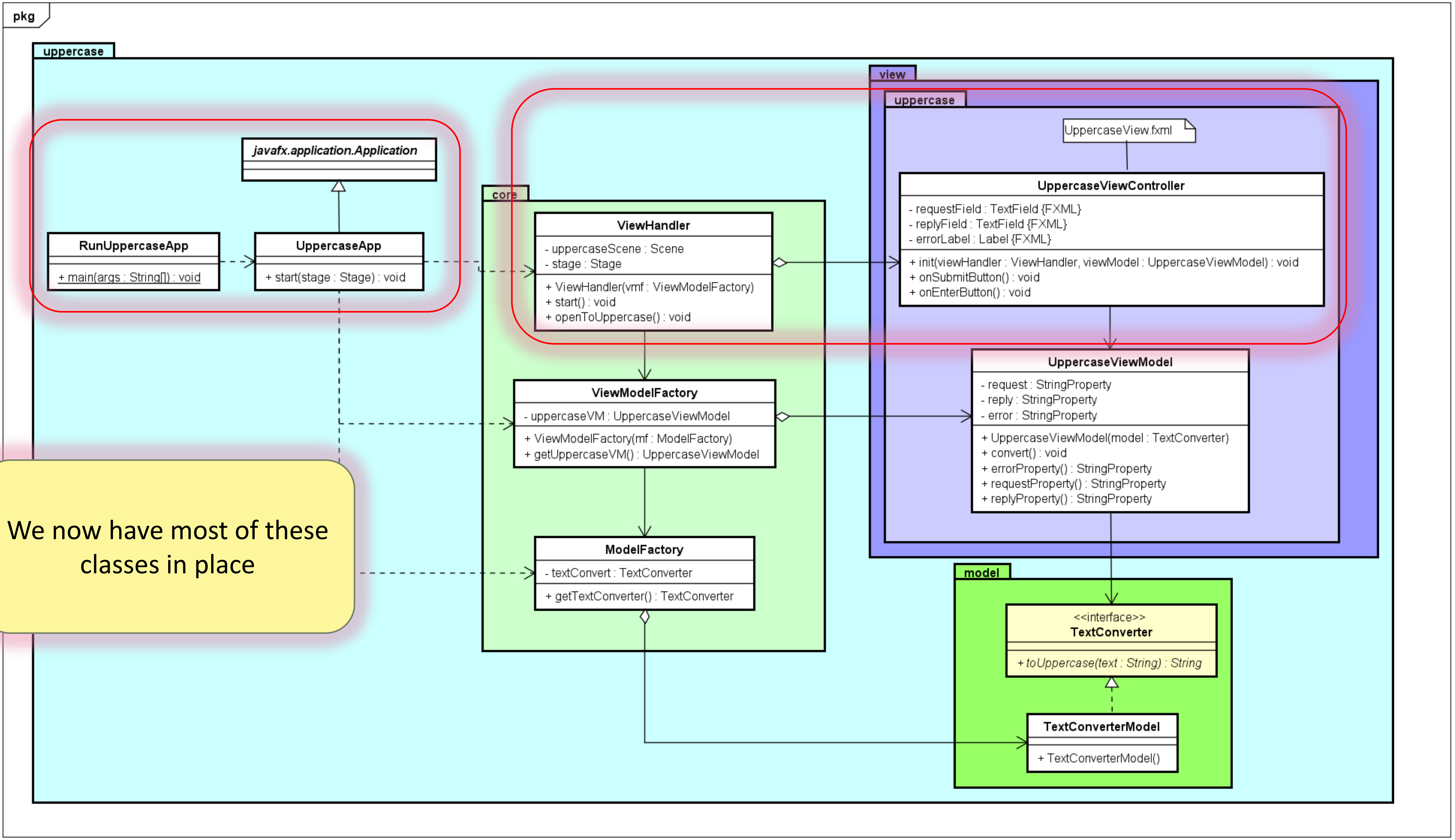
```java
public class UppercaseApp extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        ViewHandler vh = new ViewHandler();
        vh.start();
    }
}
```

- The last step before running the program is to create a class in *uppercase* package, called RunUppercaseApp

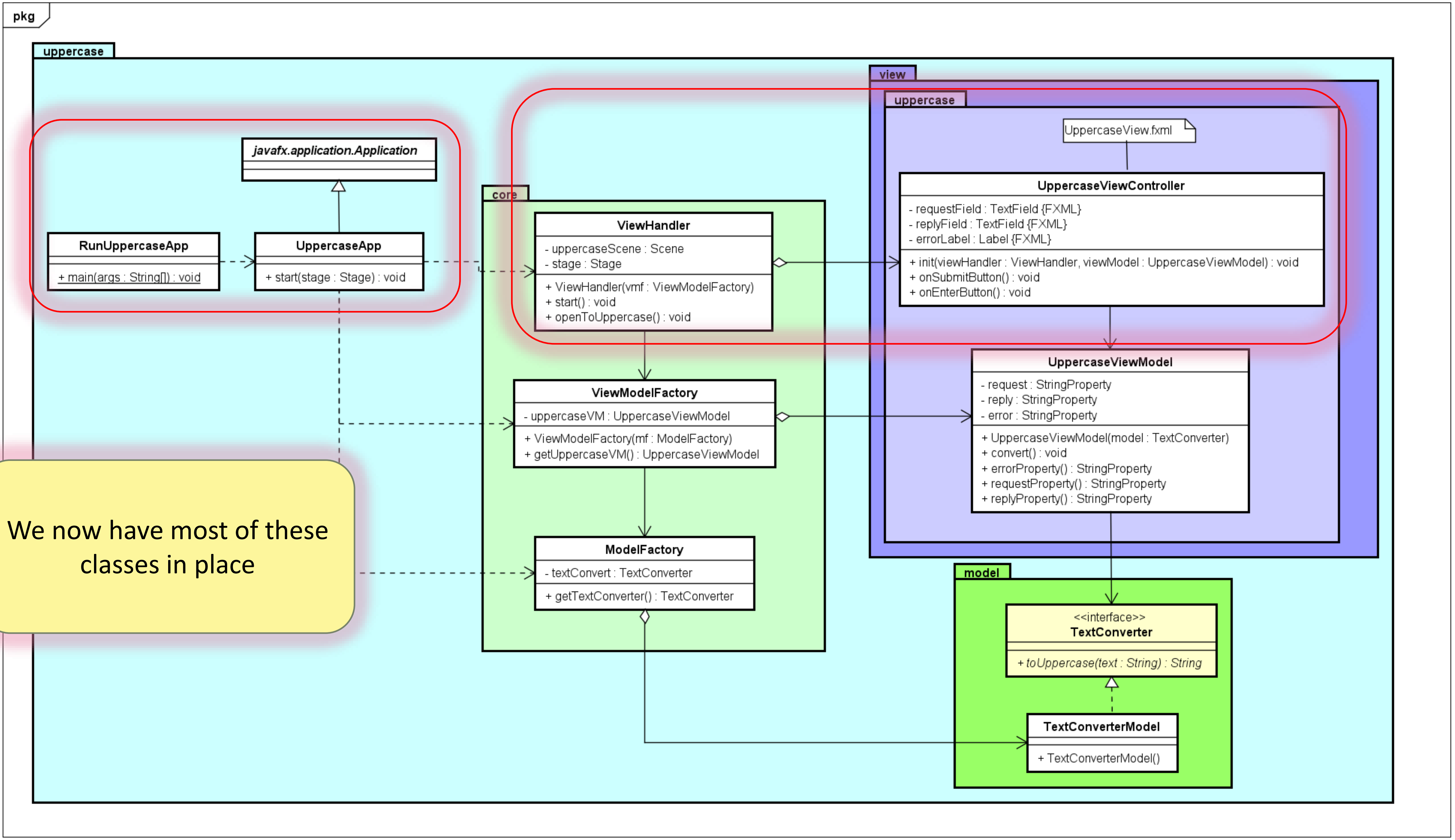
```java
public class RunUppercaseApp {

    public static void main(String[] args) {
        Application.launch(UppercaseApp.class);
    }
}
```

- Run your main method
- **You should <u>not</u> run the start method from UppercaseApp**, this will cause errors

- Your window should be open, and if you click the Submit button, you should see a print out in the console.


pkg
uppercase
javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

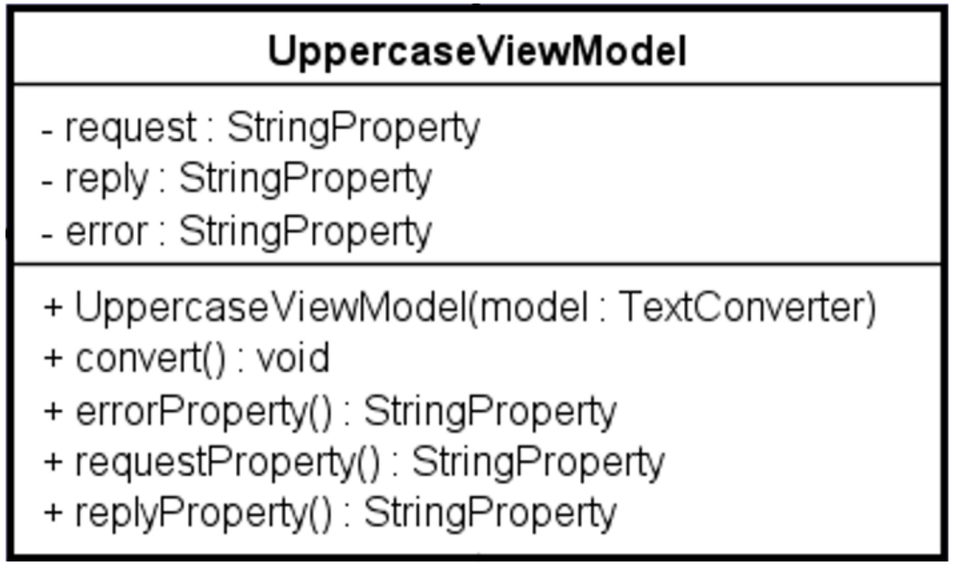
We now have most of these classes in place


pkg
uppercase
javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

We now have most of these classes in place

- The next step is to continue our top down approach. The view is sort of working, but there is little functionality behind it.
- We have the View layer more or less in place, so we will continue to the layer below: ViewModel layer.
- So, we will add the ViewModel, which will be the connection to the model.

- In package uppercase.view.uppercase create a new class called UppercaseViewModel
- Create the properties.
- Create the constructor.
  - **For now, leave out the argument.**
  - Instantiate the StringProperties to SimpleStringProperty
- Create the *convert()* method, put a print out here: “Hello from VM”, for testing purposes.
- Create the three get-property methods, which just returns their respective StringProperty.


UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty

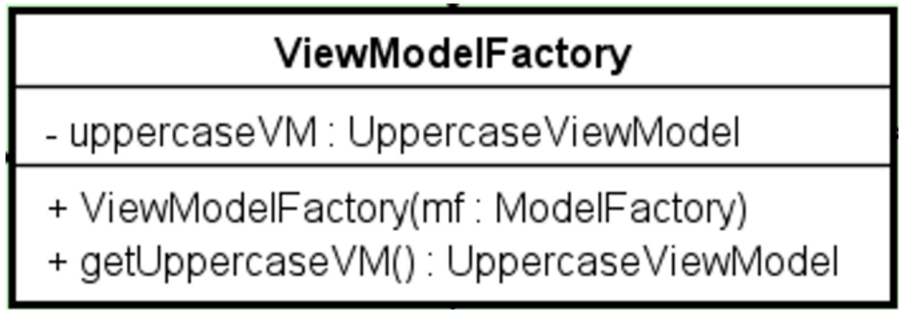
Our StringProperties, which can be bound to properties from the Controller class

Instantiating them to be SimpleStringProperties. The StringProperty is just an Interface.

For now, just print out, for testing purposes.

Get methods for the properties, so that the controller can access them

```
public class UppercaseViewModel {
    private StringProperty request, reply, error;

    public UppercaseViewModel() {
        request = new SimpleStringProperty();
        reply = new SimpleStringProperty();
        error = new SimpleStringProperty();
    }

    public void convert() {
        System.out.println("Hello from VM");
    }

    public StringProperty requestProperty() {
        return request;
    }

    public StringProperty replyProperty() {
        return reply;
    }

    public StringProperty errorProperty() {
        return error;
    }
}
```


ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel

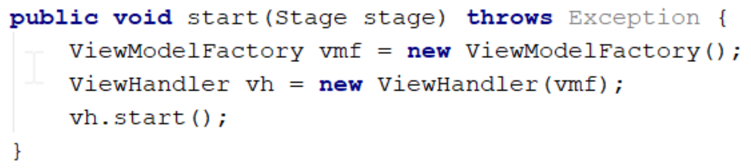
- We now need a factory for the ViewModel.
- In the core package, create the ViewModelFactory.
- Do not include the constructor argument for now.
- Implement the *getUppercaseVM()* method so that it first checks if the uppercaseVM is null, and if so, it instantiates it, and afterwards returns it. We use lazy instantiation here, meaning we only create an object, when we need it. This is also done, so that the ViewModel may be reused, if needed.

Empty constructor for now, we will expand it later.

Lazy instantiation and reusing the instance. If this method is called multiple times, the same instance is returned every time, instead of creating a new one.

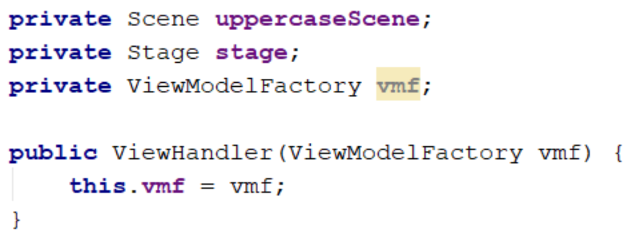
```java
public class ViewModelFactory {

    private UppercaseViewModel uppercaseViewModel;

    public ViewModelFactory() {
    }

    public UppercaseViewModel getUppercaseViewModel() {
        if (uppercaseViewModel == null)
            uppercaseViewModel = new UppercaseViewModel();
        return uppercaseViewModel;
    }
}
```

- Go to UppercaseApp.
- Instantiate a ViewModelFactory and pass it to the constructor of ViewHandler.



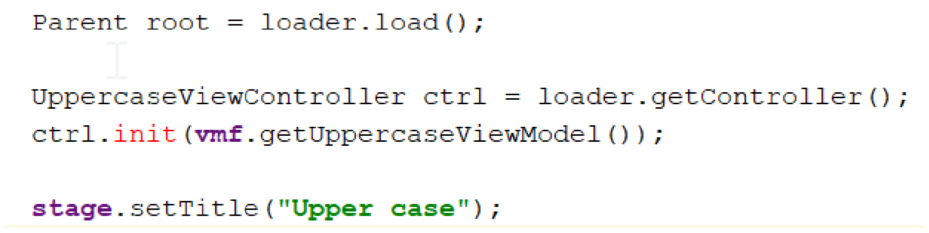
```
public void start(Stage stage) throws Exception {
    ViewModelFactory vmf = new ViewModelFactory();
    ViewHandler vh = new ViewHandler(vmf);
    vh.start();
}
```

- IntelliJ will complain, so create an appropriate constructor. In here, assign the ViewModelFactory to a field variable inside ViewHandler.



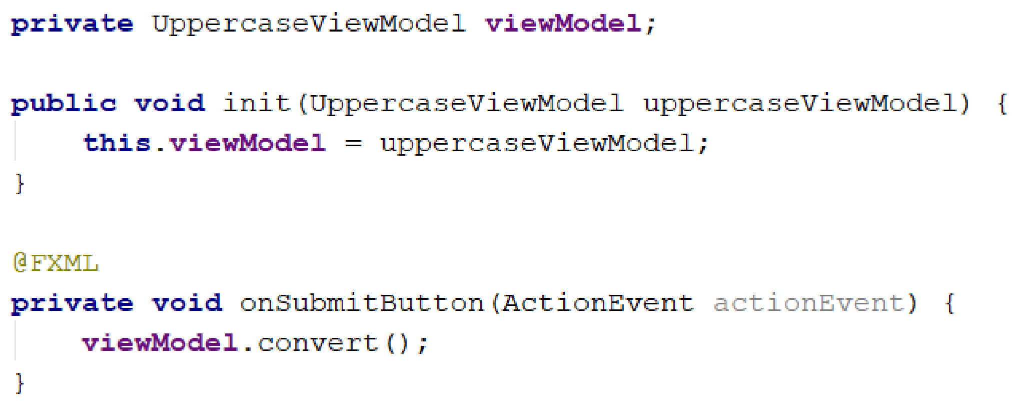
```
private Scene uppercaseScene;
private Stage stage;
private ViewModelFactory vmf;

public ViewHandler(ViewModelFactory vmf) {
    this.vmf = vmf;
}
```

- In the method ViewHandler::openToUppercase, do:
  1. After *.load()* method is called, insert:
  2. Create the *init()* in UppercaseViewController. You can alt+enter on the init here.



```
Parent root = loader.load();

UppercaseViewController ctrl = loader.getController();
ctrl.init(vmf.getUppercaseViewModel());

stage.setTitle("Upper case");
```

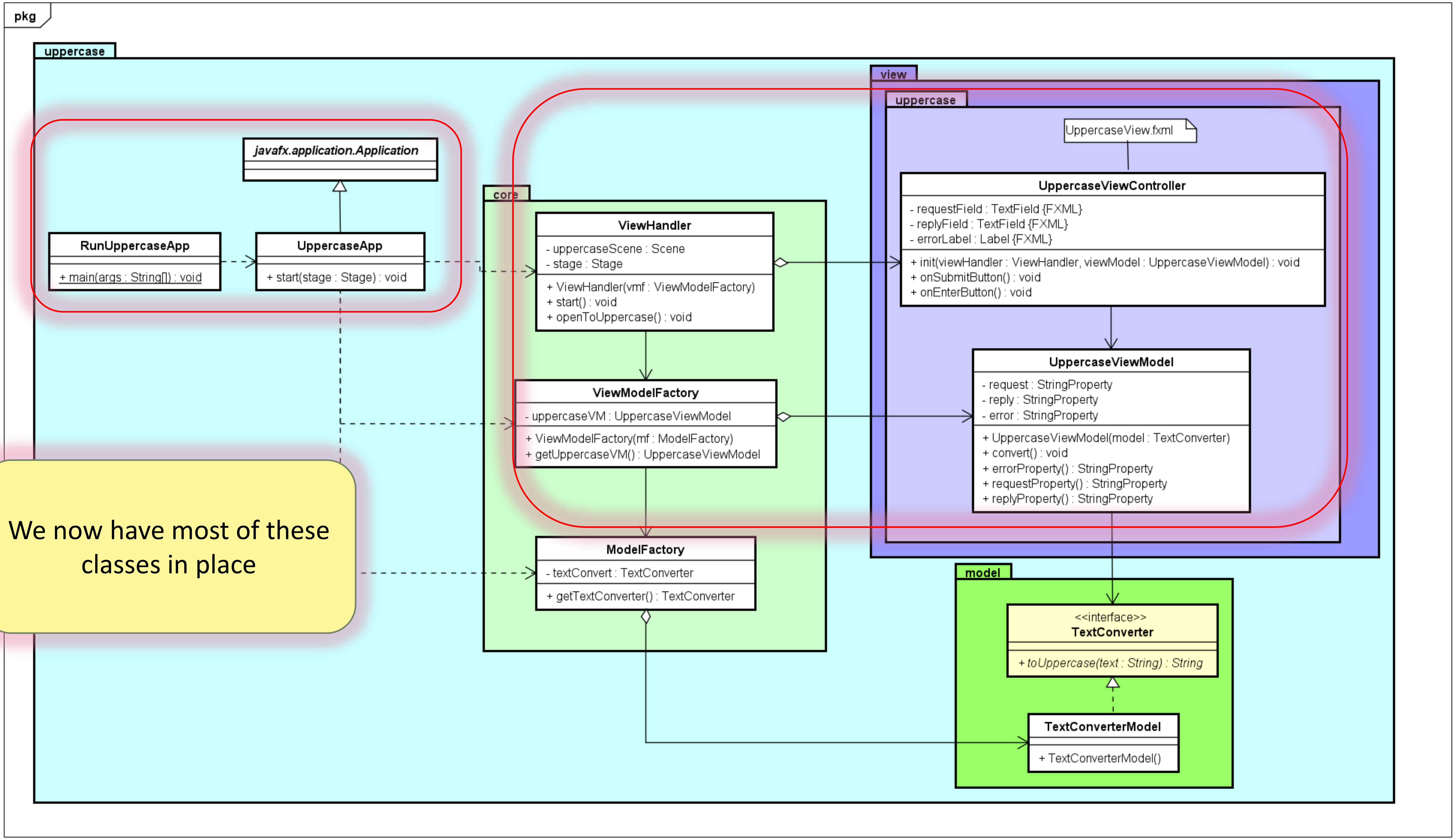
- In UppercaseViewController assign the constructor argument to a field variable.
- In UppercaseViewController::onSubmitButton call viewModel.convert()



```
private UppercaseViewModel viewModel;

public void init(UppercaseViewModel uppercaseViewModel) {
    this.viewModel = uppercaseViewModel;
}

@FXML
private void onSubmitButton(ActionEvent actionEvent) {
    viewModel.convert();
}
```

- Now, the ViewHandler knows about the ViewModelFactory, and can therefore initialize our controller correctly.
- The UppercaseViewController knows about its view model and can call the functionality on this class.

- Run your main method, and verify that you see the “Hello from VM” printed out.



pkg
uppercase
javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

We now have most of these classes in place

- The next step is to finish the connection between UppercaseViewController and UppercaseViewModel.
- This means we must bind the properties.
- In your UppercaseViewController *init()* method, make the bindings:
  - errorLabel – bind onedirectional to error
  - requestField – bind bidirectional to request
  - replyField – bind onedirectional to reply

```java
public void init(UppercaseViewModel uppercaseViewModel) {
    this.viewModel = uppercaseViewModel;
    errorLabel.textProperty().bind(viewModel.errorProperty());
    requestField.textProperty().bindBidirectional(viewModel.requestProperty());
    replyField.textProperty().bind(viewModel.replyProperty());
}
```

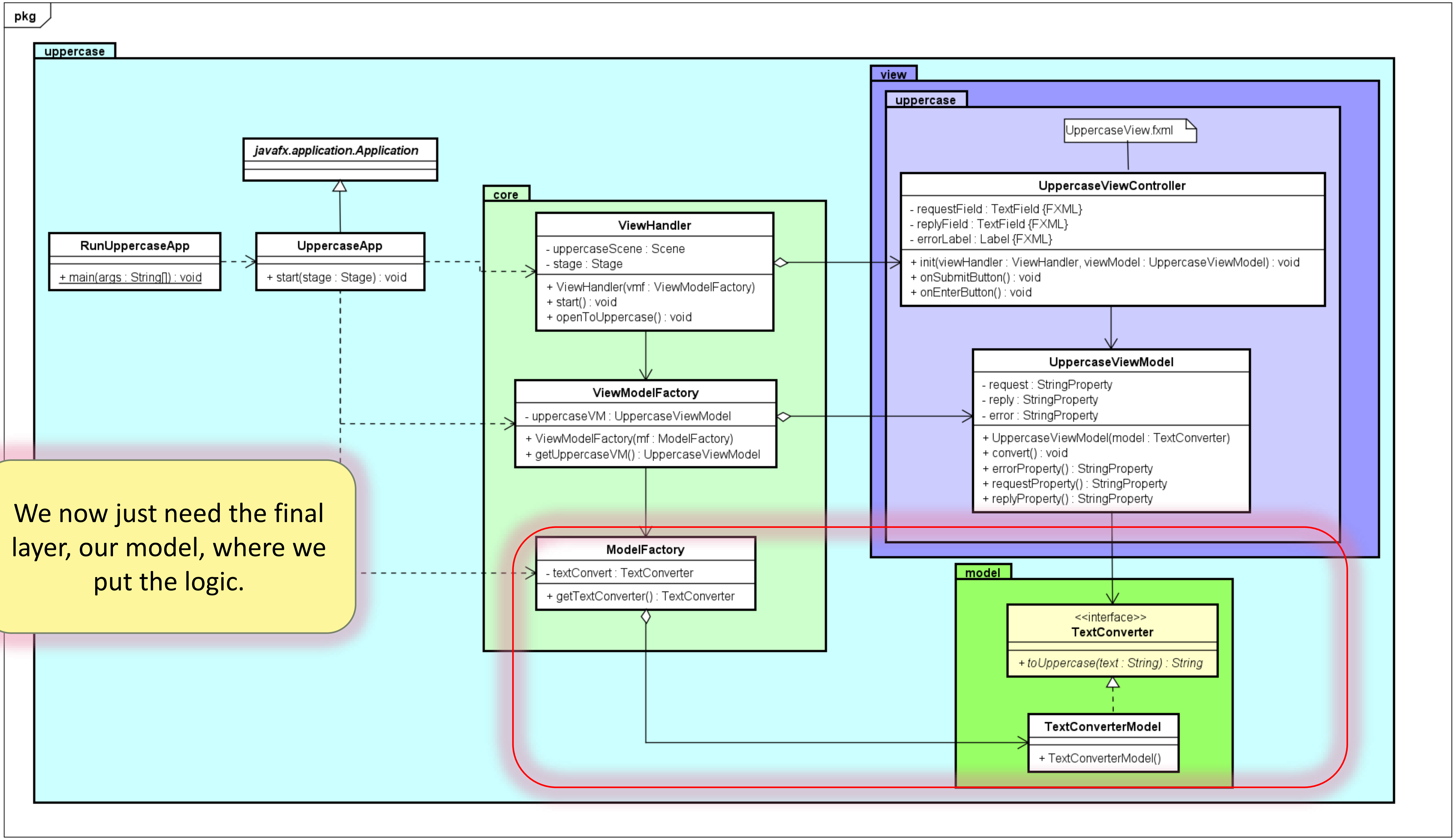
Error labels can only listen to other properties, so it’s one directional

We use bidirectional because the user input must be pushed to the VM, and the VM must be able to clear the TextField. Data flows both ways

We use onedirectional, because the user cannot input into this field. The field should be greyed out. Data only flows from VM to View here.

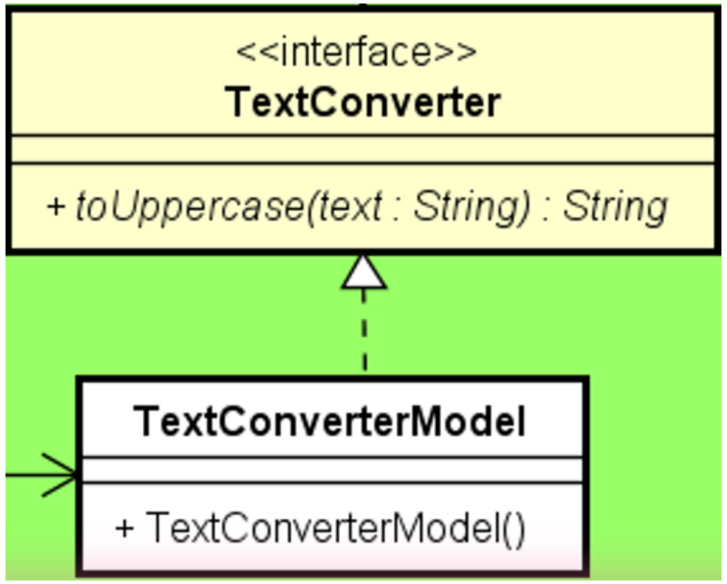
- In the UppercaseViewModel change the *convert()* method to print out the data from the request property.
- Also in *convert(),* insert an if-check to see if the data in request is empty. If so, update the error property with an appropriate message. We perform input validation.

- Run your main method, insert text into the text field, and check the console for prints, when you click Submit.
- Also check that if you click Submit with an empty input field, you see the error.


pkg
uppercase
javafx.application.Application
RunUppercaseApp
+ main(args : String[]) : void
UppercaseApp
+ start(stage : Stage) : void
core
ViewHandler
- uppercaseScene : Scene
- stage : Stage
+ ViewHandler(vmf : ViewModelFactory)
+ start() : void
+ openToUppercase() : void
ViewModelFactory
- uppercaseVM : UppercaseViewModel
+ ViewModelFactory(mf : ModelFactory)
+ getUppercaseVM() : UppercaseViewModel
ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter
view
uppercase
UppercaseView.fxml
UppercaseViewController
- requestField : TextField {FXML}
- replyField : TextField {FXML}
- errorLabel : Label {FXML}
+ init(viewHandler : ViewHandler, viewModel : UppercaseViewModel) : void
+ onSubmitButton() : void
+ onEnterButton() : void
UppercaseViewModel
- request : StringProperty
- reply : StringProperty
- error : StringProperty
+ UppercaseViewModel(model : TextConverter)
+ convert() : void
+ errorProperty() : StringProperty
+ requestProperty() : StringProperty
+ replyProperty() : StringProperty
model
<<interface>>
TextConverter
+ toUppercase(text : String) : String
TextConverterModel
+ TextConverterModel()

We now just need the final layer, our model, where we put the logic.

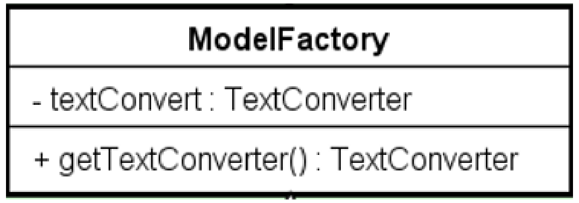
- Create the interface in <u>model</u> package
- In the same package, create
  - It should implement TextConverter
  - Override *toUppercase()*, in here convert the argument to uppercase, using String class' *toUppercase()* method.
  - Return the result.



```java
public interface TextConverter {

    String toUppercase(String text);

}
```

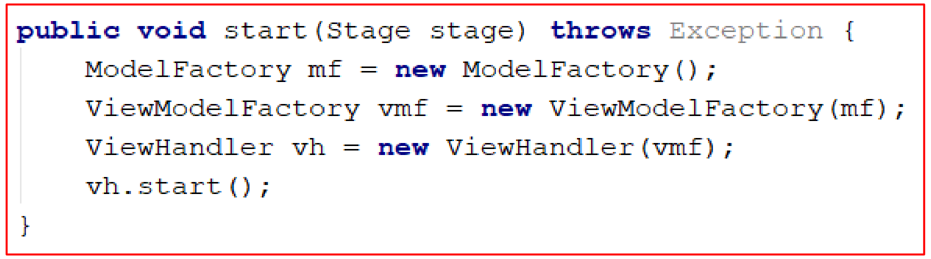
```java
public class TextConverterModel implements TextConverter {

    @Override
    public String toUppercase(String text) {
        return text.toUpperCase();
    }

}
```

- In core package, create a ModelFactory.
- No explicit constructor is needed
- The get-method uses lazy instantiation, similar to the ViewModelFactory
- This enables sharing of the same TextConverter instance.
- Often, model instances are shared between multiple view models

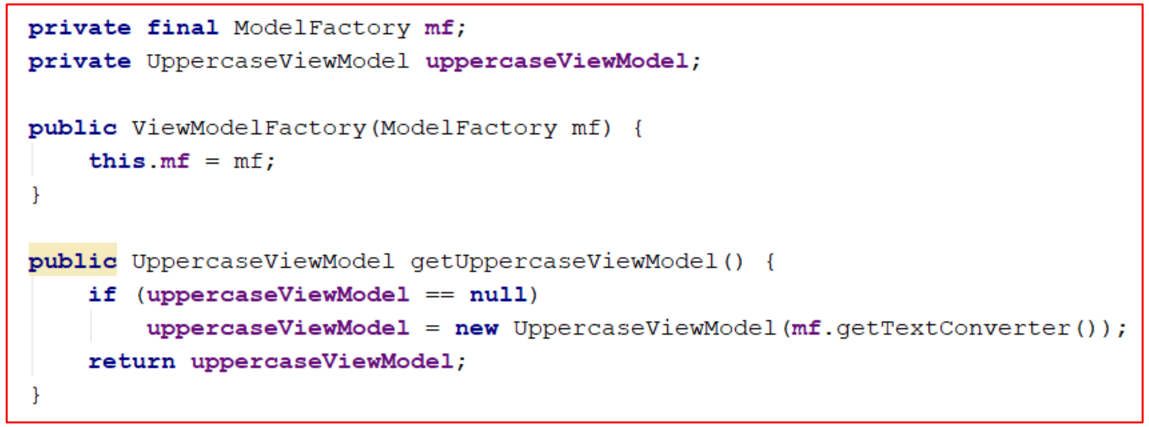

ModelFactory
- textConvert : TextConverter
+ getTextConverter() : TextConverter

```java
public class ModelFactory {

    private TextConverter textConverter;

    public TextConverter getTextConverter() {
        if(textConverter == null)
            textConverter = new TextConverterModel();
        return textConverter;
    }
}
```

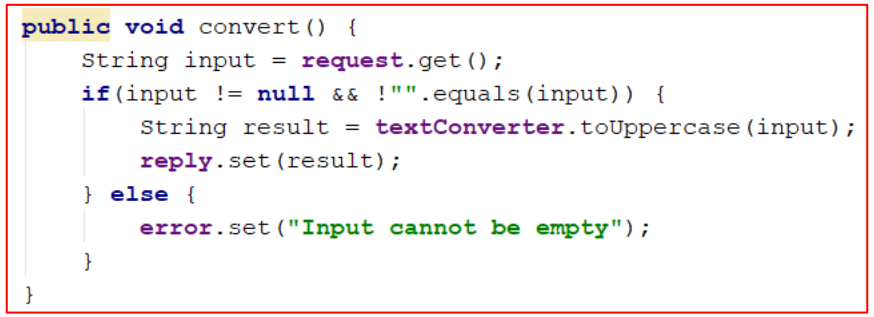
- In UppercaseApp::start instantiate a ModelFactory, pass it to ViewModelFactory through its constructor, and assign it to a field variable in the ViewModelFactory (the constructor must be updated)
- Now, when you instantiate the UppercaseViewModel, pass it an instance of TextConverter from the ModelFactory (the constructor must be updated).
- Store the TextConverter in a field variable in UppercaseViewModel
- In UppercaseViewModel::convert call TextConverter::toUppercase, and put the result into reply String property.



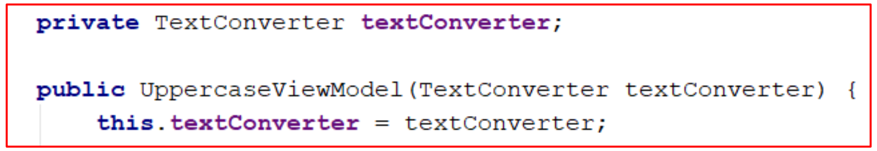
```java
public void start(Stage stage) throws Exception {
    ModelFactory mf = new ModelFactory();
    ViewModelFactory vmf = new ViewModelFactory(mf);
    ViewHandler vh = new ViewHandler(vmf);
    vh.start();
}
```



```java
private final ModelFactory mf;
private UppercaseViewModel uppercaseViewModel;

public ViewModelFactory(ModelFactory mf) {
    this.mf = mf;
}

public UppercaseViewModel getUppercaseViewModel() {
    if (uppercaseViewModel == null)
        uppercaseViewModel = new UppercaseViewModel(mf.getTextConverter());
    return uppercaseViewModel;
}
```



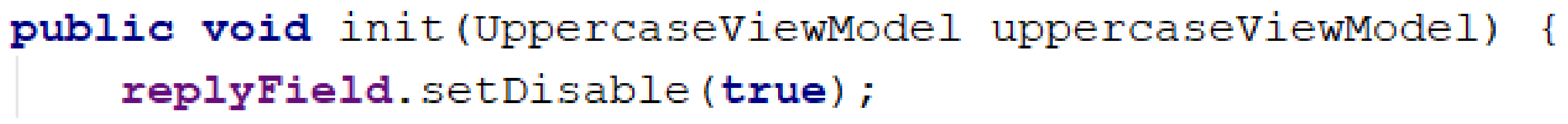
```java
public void convert() {
    String input = request.get();
    if(input != null && !"".equals(input)) {
        String result = textConverter.toUppercase(input);
        reply.set(result);
    } else {
        error.set("Input cannot be empty");
    }
}
```



```java
private TextConverter textConverter;

public UppercaseViewModel(TextConverter textConverter) {
    this.textConverter = textConverter;
```

- Run your main method.
- Test that the functionality works as expected, both with empty input and valid input.

- Currently you can still click in the reply field, though you cannot write.
- If you want it disabled, i.e. greyed out and unclickable, you can do this in UppercaseViewController:



```java
public void init(UppercaseViewModel uppercaseViewModel) {
    replyField.setDisable(true);
```

- This concludes the tutorial

- We used a top-down approach, we start with showing a view, though without functionality.
- This is often a good approach, because for each step, or layer, we add, we can run the program and verify that the classes are connected correctly, by printing out messages along the way.
- MVVM can quickly become a bit confusing, and connections between classes are easy to forget.

- Finally, the source code can be found here:
- https://github.com/TroelsMortensen/UppercaseMVVM