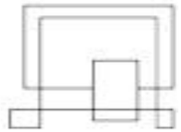
Life is great
VIA University College



Software Development with UML and Java 2

Model, View, View-model (MVVM) & JavaFX

Part 1

Agenda

- MVVM – Model, View, View model
- Motivation
- Concept
- Responsibilities of the different parts

- Part 2: Connection and communication between the different parts
- Part 3: UML template
- Part 4: Examples and code

Dividing responsibilities

- Architectural pattern
- Similar approaches:
  - MVC: Model, View, Controller
  - MVP: Model, View, Presenter

- MVVM, how to:
  - structure your GUI
  - Structure your Data/Business Logic
  - communicate with each other.

- MVVM is used across multiple platforms and languages:
  - .NET, Android, JavaFX

Dividing responsibilities

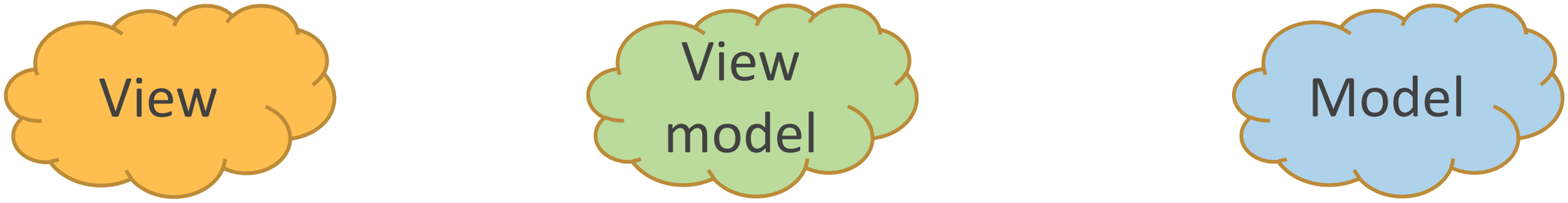
- If you’re not careful, you might just end up with a class that does sort of everything.
- It has GUI elements
- Lists of data
- Logic, like list-filtering, or sorting, translating from GUI to data in e.g. a file.

Dividing responsibilities

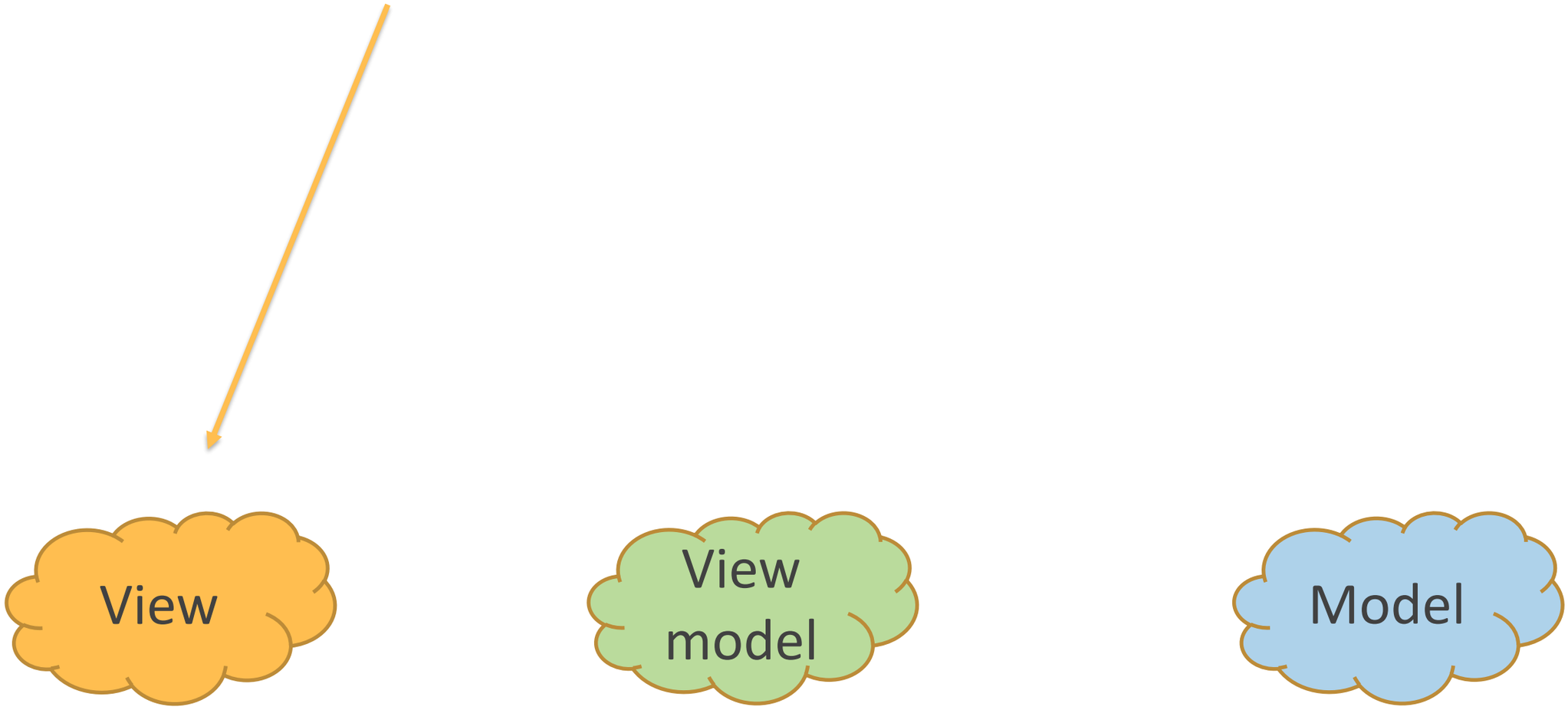
- When everything is bundled together, it's difficult to:
  - Test a feature, you'll have to start the program, insert relevant stuff, click different places in the GUI. We're going to look at "automated" tests later in the semester.
  - Maintain; you don't have a clear picture of where stuff happens, and where to put new stuff.
  - Add new functionality without modifying existing classes. We really want to avoid meddling with already working stuff.
  - If something breaks, it can be difficult to find.

MVVM in short

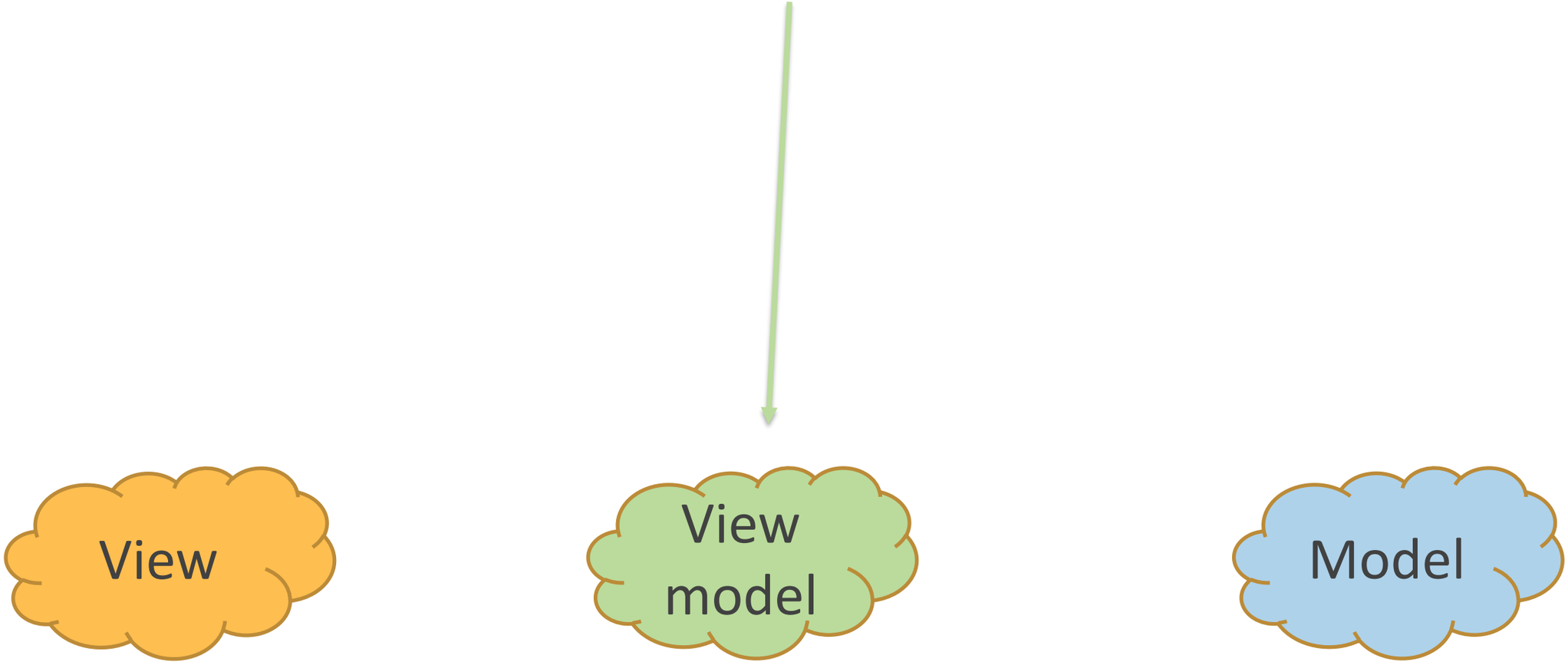
- Says something about how to structure a program, which will display some data.
- Says nothing about client server systems, we'll get to this later


View
View model
Model

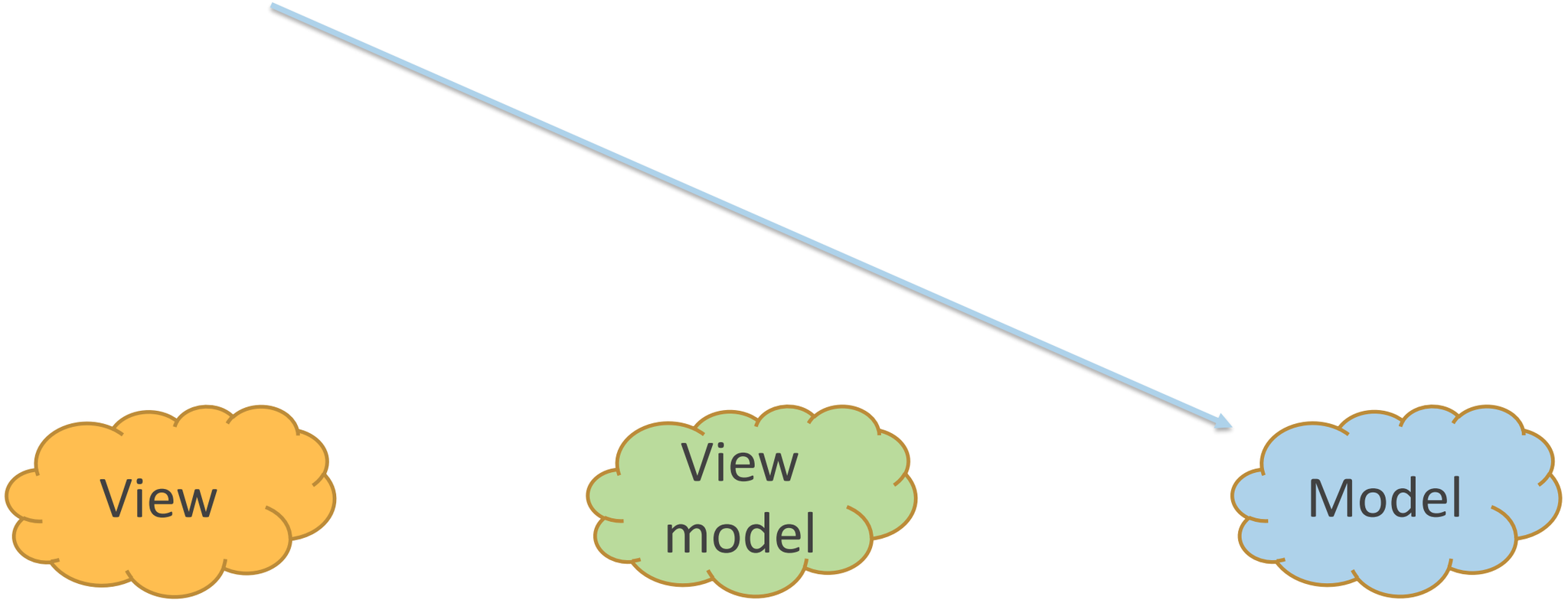
- You can have many GUI views



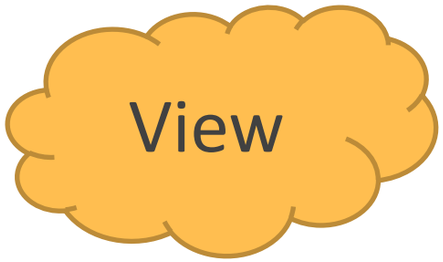
- You can have many GUI views
- Each view will typically have a corresponding View model class

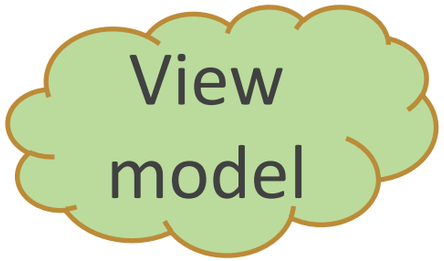


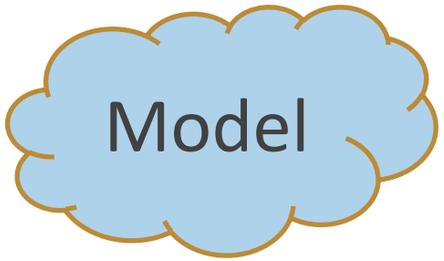
- You can have many GUI views
- Each view will typically have a corresponding View model class
- The Model can be any number of classes, grouped together, used by the View models



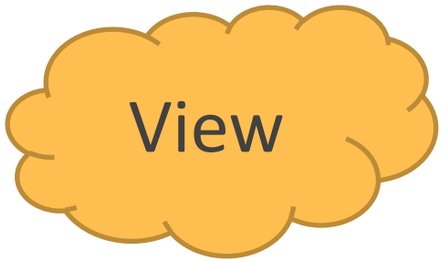
- Where do we put stuff?

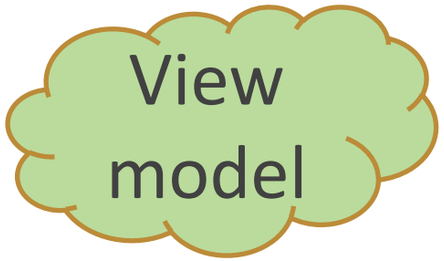


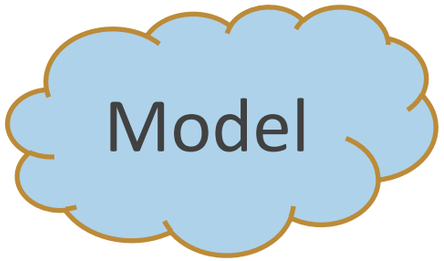




Presentation	Take user input	User event handling	Data	Business logic (data manipulation)	GUI state
Storage/ data base	Rendering	Translation	Parsing (XML, JSON)	Formatting	Networking







Take user input

User event handling

Data

Business logic (data manipulation)

GUI state

Storage/ data base

Rendering

Translation

Parsing (XML, JSON)

Formatting

Networking

Presentation

Showing stuff to the user: Data, images, graphs, information

View

View model

Model

Take user input

User event handling

Data

Business logic (data manipulation)

GUI state

Storage/ data base

Rendering

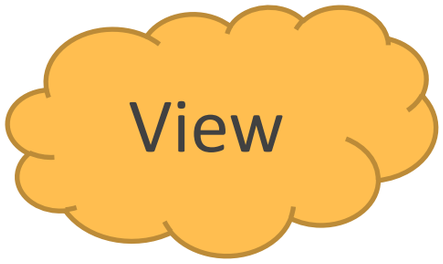
Translation

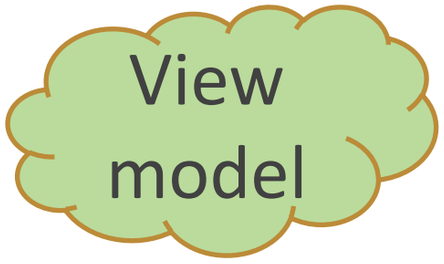
Parsing (XML, JSON)

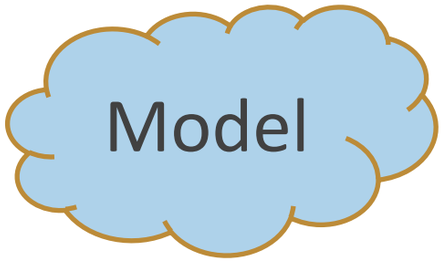
Formatting

Networking

Presentation







User event handling

Data

Business logic (data manipulation)

GUI state

Storage/ data base

Rendering

Translation

Parsing (XML, JSON)

Formatting

Networking

Take user input

Buttons, radio buttons, check boxes, textfields. Any way of interacting with the system

Presentation

View

View model

Model

User event handling

Data

Business logic (data manipulation)

GUI state

Storage/ data base

Rendering

Translation

Parsing (XML, JSON)

Formatting

Networking

Take user input

Presentation

View

View model

Model

Data

Business logic
(data manipulation)

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

User event
handling

Whenever a button is clicked, something has to be done

Take user input

Presentation

View

View model

Model

Data

Business logic
(data manipulation)

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

Take user input

Presentation

View

User event
handling

View
model

Model

Business logic
(data manipulation)

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

Data

Any kind of data or
information

Take user input

Presentation

View

User event
handling

View
model

Model

Business logic
(data manipulation)

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

Take user input

Presentation

View

User event
handling

View
model

Data

Model

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

Business logic
(data manipulation)

Changing, modifying data.
Calculations, rules,
validation.

Take user input

Presentation

View

User event
handling

View
model

Data

Model

GUI state

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

Take user input

Presentation

View

User event
handling

View
model

Business logic
(data manipulation)

Data

Model

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Formatting

Networking

GUI state

Is a button enabled or disabled?
Is a radio button selected?
Which element in a list is selected?

Take user input

Presentation

View

User event
handling

View
model

Business logic
(data manipulation)

Data

Model

Storage/
data base

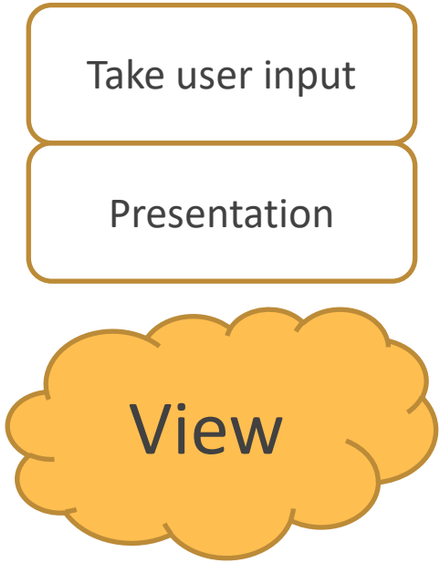
Rendering

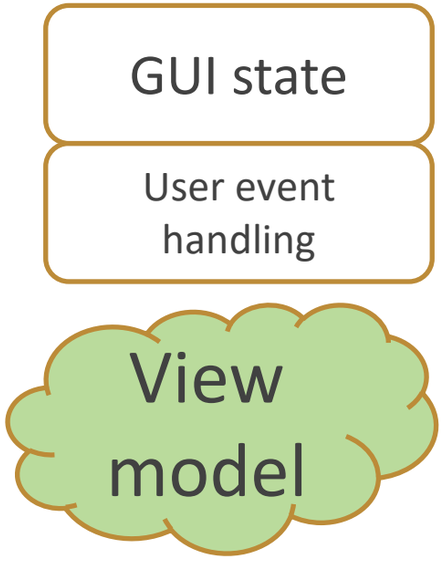
Translation

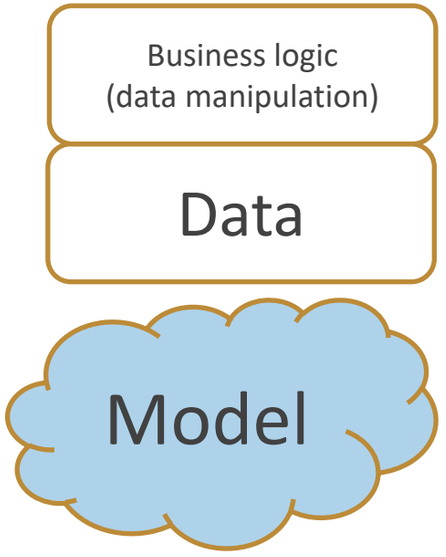
Parsing (XML,
JSON)

Formatting

Networking







Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Networking

Formatting

Data can be held in one form, e.g. a
double for currency.
But the GUI will show it as a String

Take user input

Presentation

View

GUI state

User event
handling

View
model

Business logic
(data manipulation)

Data

Model

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Networking

Formatting

A Person class with information will be an object in the Model.
In the GUI the Person data is put into different text-fields, checkboxes, etc.

Take user input

Presentation

View

GUI state

User event handling

View model

Business logic
(data manipulation)

Data

Model

Storage/
data base

Rendering

Translation

Parsing (XML,
JSON)

Networking

Take user input

Presentation

View

Formatting

GUI state

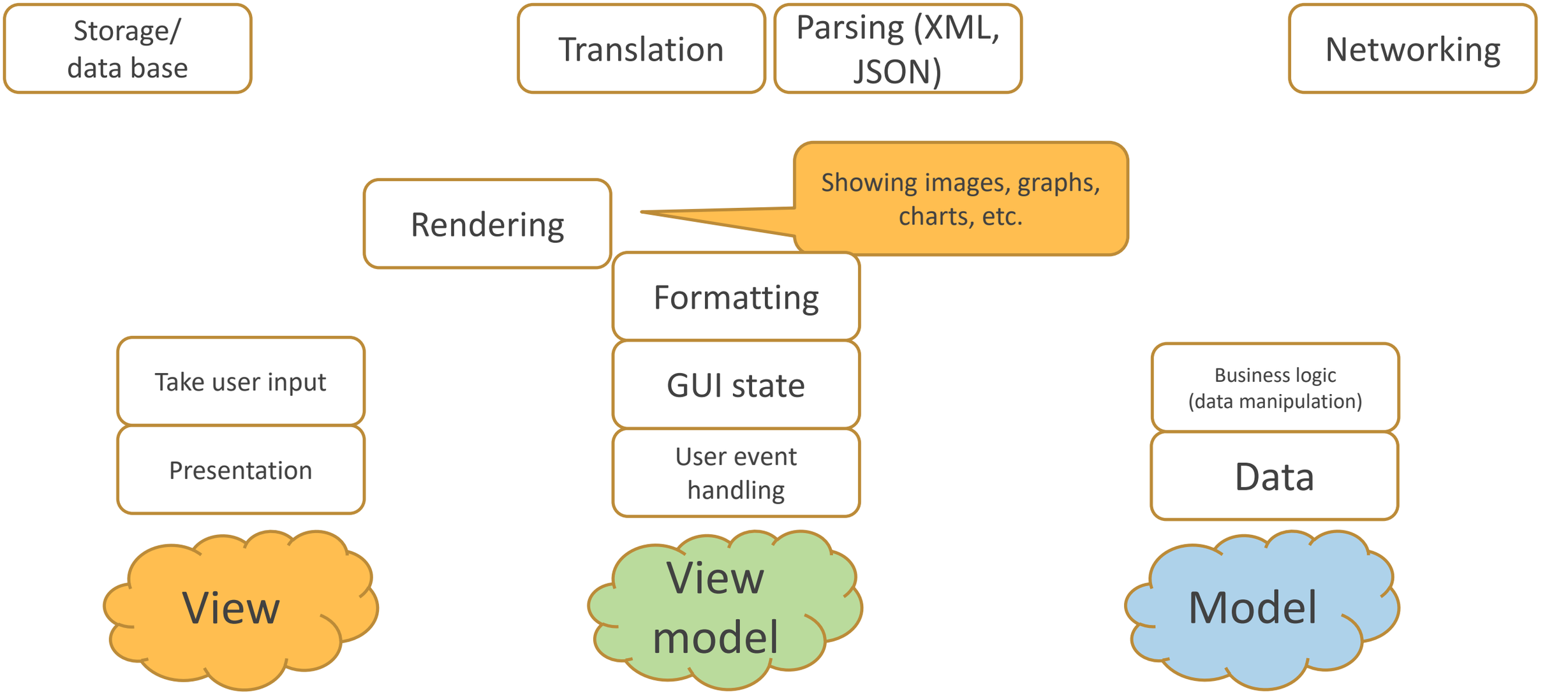
User event
handling

View
model

Business logic
(data manipulation)

Data

Model


Storage/
data base
Translation
Parsing (XML,
JSON)
Networking
Rendering
Showing images, graphs,
charts, etc.
Formatting
GUI state
User event
handling
Take user input
Presentation
View
View
model
Business logic
(data manipulation)
Data
Model

Storage/
data base

Translation

Parsing (XML,
JSON)

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event
handling

View
model

Business logic
(data manipulation)

Data

Model

Storage/
data base

Translation

Networking

Parsing (XML, JSON)

You may receive XML or JSON from somewhere else, and it must be translated into objects

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

View model

Business logic
(data manipulation)

Data

Model

Storage/
data base

Translation

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event
handling

View
model

Parsing (XML,
JSON)

Business logic
(data manipulation)

Data

Model

Storage/
data base

Accessing a database, or other kind of persistent storage

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

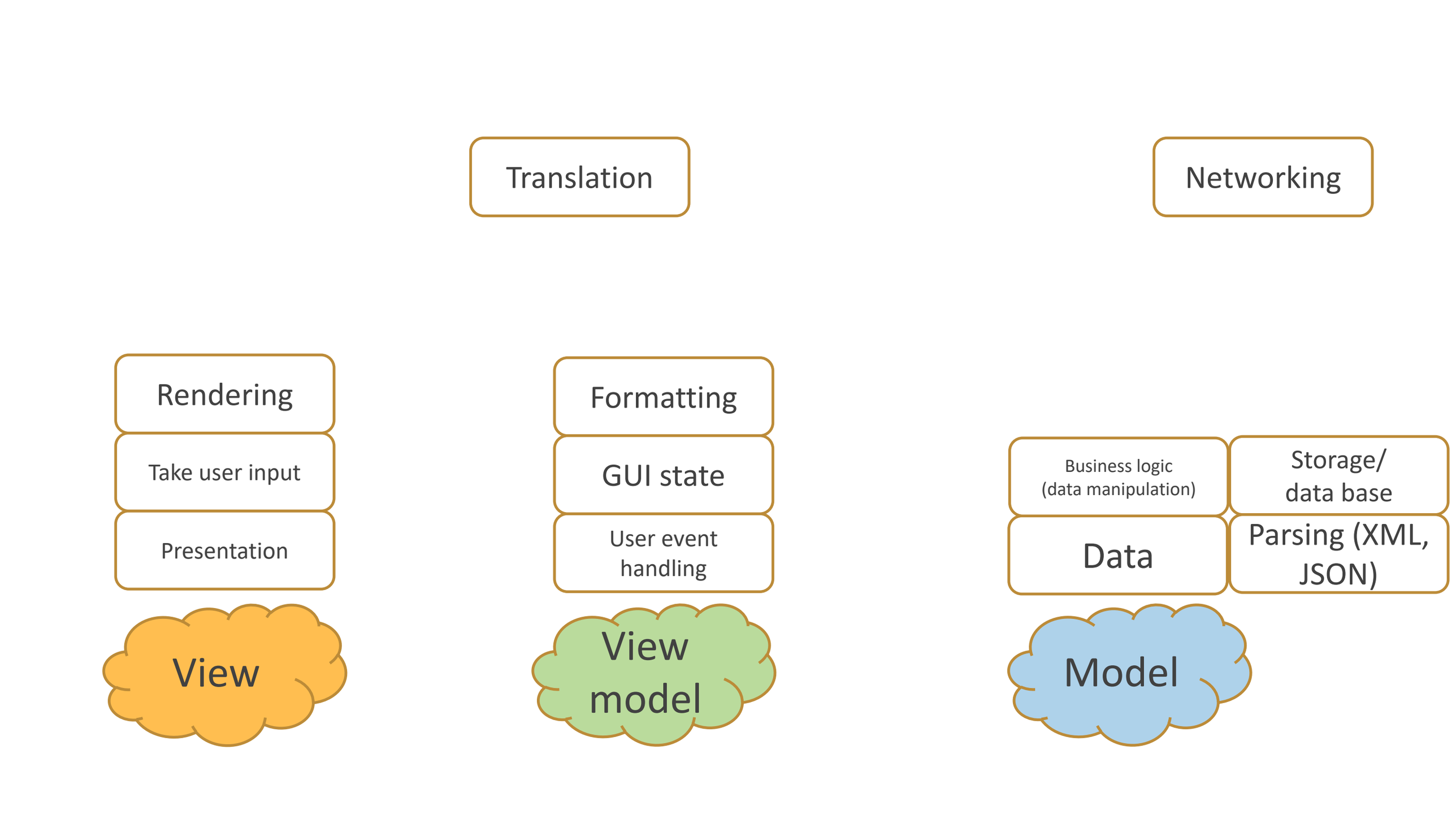
View model

Parsing (XML, JSON)

Business logic (data manipulation)

Data

Model


Translation
Networking
Rendering
Take user input
Presentation
View
Formatting
GUI state
User event handling
View model
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
Model

Between languages, or from one valuta to another. From JSON to XML.

Translation

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

View model

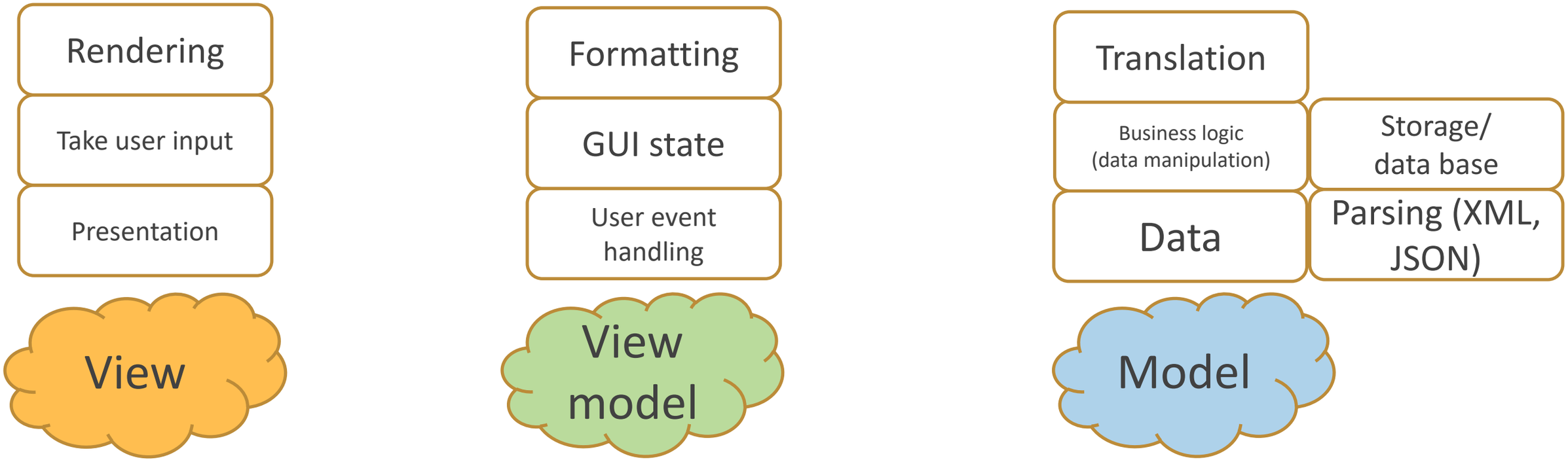
Business logic (data manipulation)

Storage/ data base

Data

Parsing (XML, JSON)

Model


Networking
Rendering
Take user input
Presentation
View
Formatting
GUI state
User event handling
View model
Translation
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
Model

Connecting a client program to a server program

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

View model

Translation

Business logic (data manipulation)

Storage/ data base

Data

Parsing (XML, JSON)

Model

Sometimes a part of the Model.
Often a separate “cloud”

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

View model

Translation

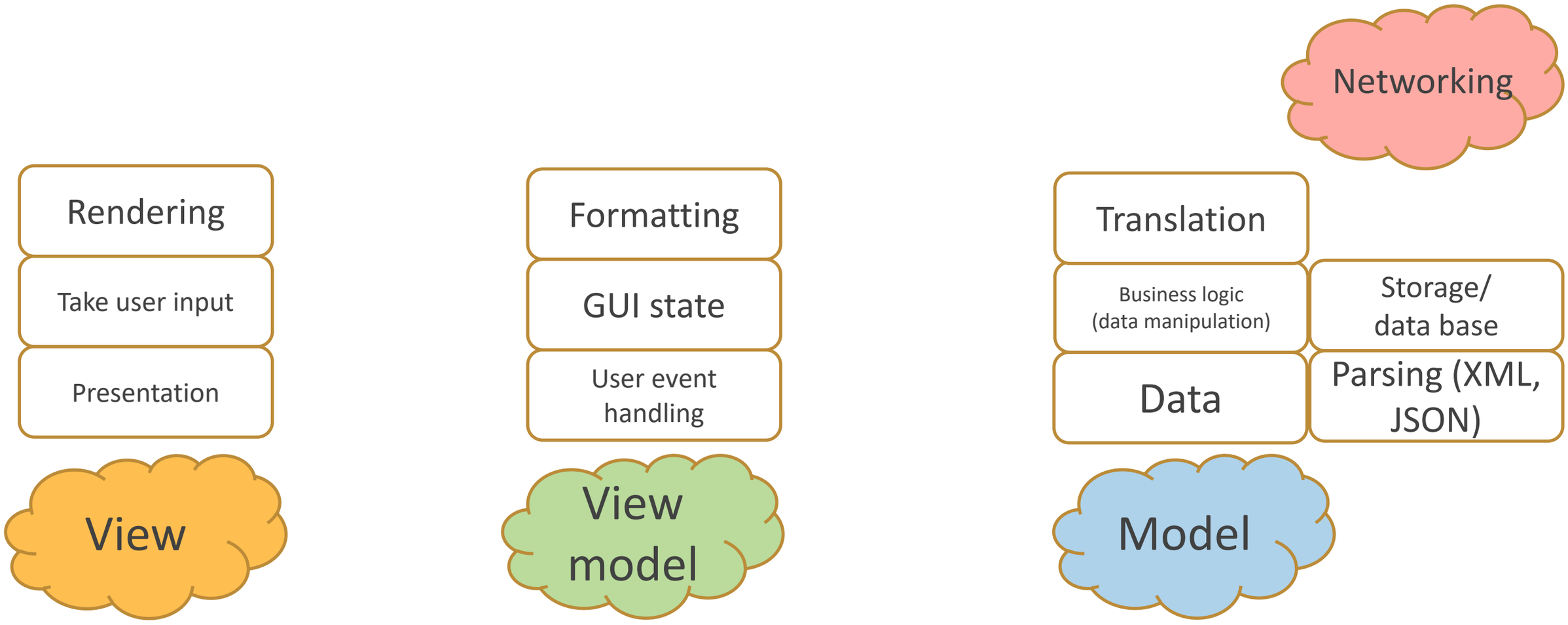
Business logic (data manipulation)

Storage/ data base

Data

Parsing (XML, JSON)

Model


Networking
Rendering
Take user input
Presentation
Formatting
GUI state
User event handling
Translation
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
View
View model
Model

Input validation

Some simple input validation.
Is the input actually a number?
Is it in the general form of an email?
Is it a phone number?

Networking

Rendering

Take user input

Presentation

View

Formatting

GUI state

User event handling

View model

Translation

Business logic (data manipulation)

Storage/ data base

Data

Parsing (XML, JSON)

Model

Data validation

Does the email already exist?
Does the number actually make sense?
Does the name contain numbers?

Networking

Rendering

Take user input

Presentation

View

Input validation

Formatting

GUI state

User event handling

View model

Translation

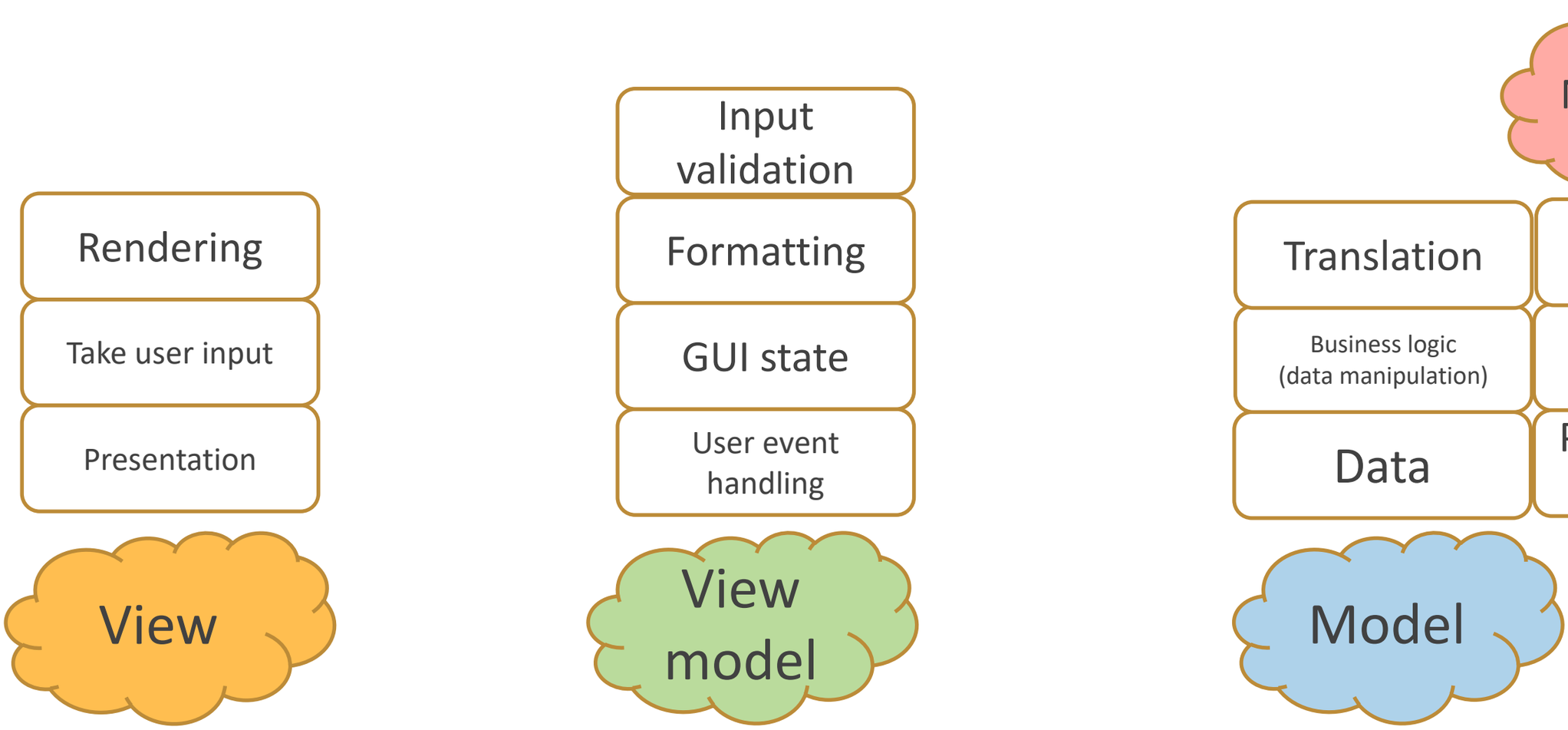
Business logic (data manipulation)

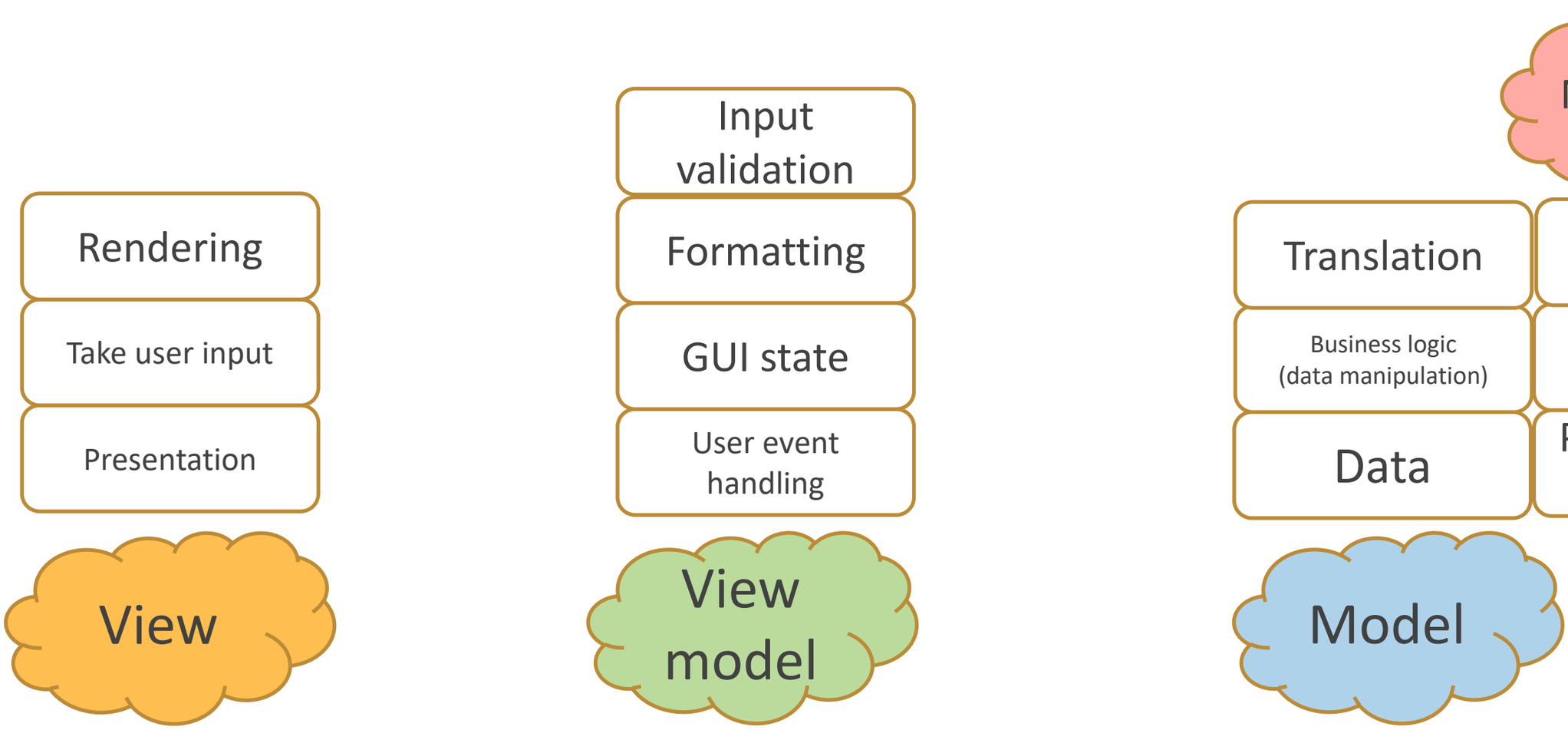
Storage/ data base

Data

Parsing (XML, JSON)

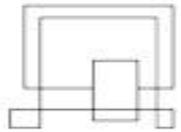
Model


Rendering
Take user input
Presentation
View
Input validation
Formatting
GUI state
User event handling
View model
Networking
Translation
Data validation
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
Model


Rendering
Take user input
Presentation
View
Input validation
Formatting
GUI state
User event handling
View model
Networking
Translation
Data validation
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
Model

End of part 1

Life is great
VIA University College



Software Development with UML and Java 2

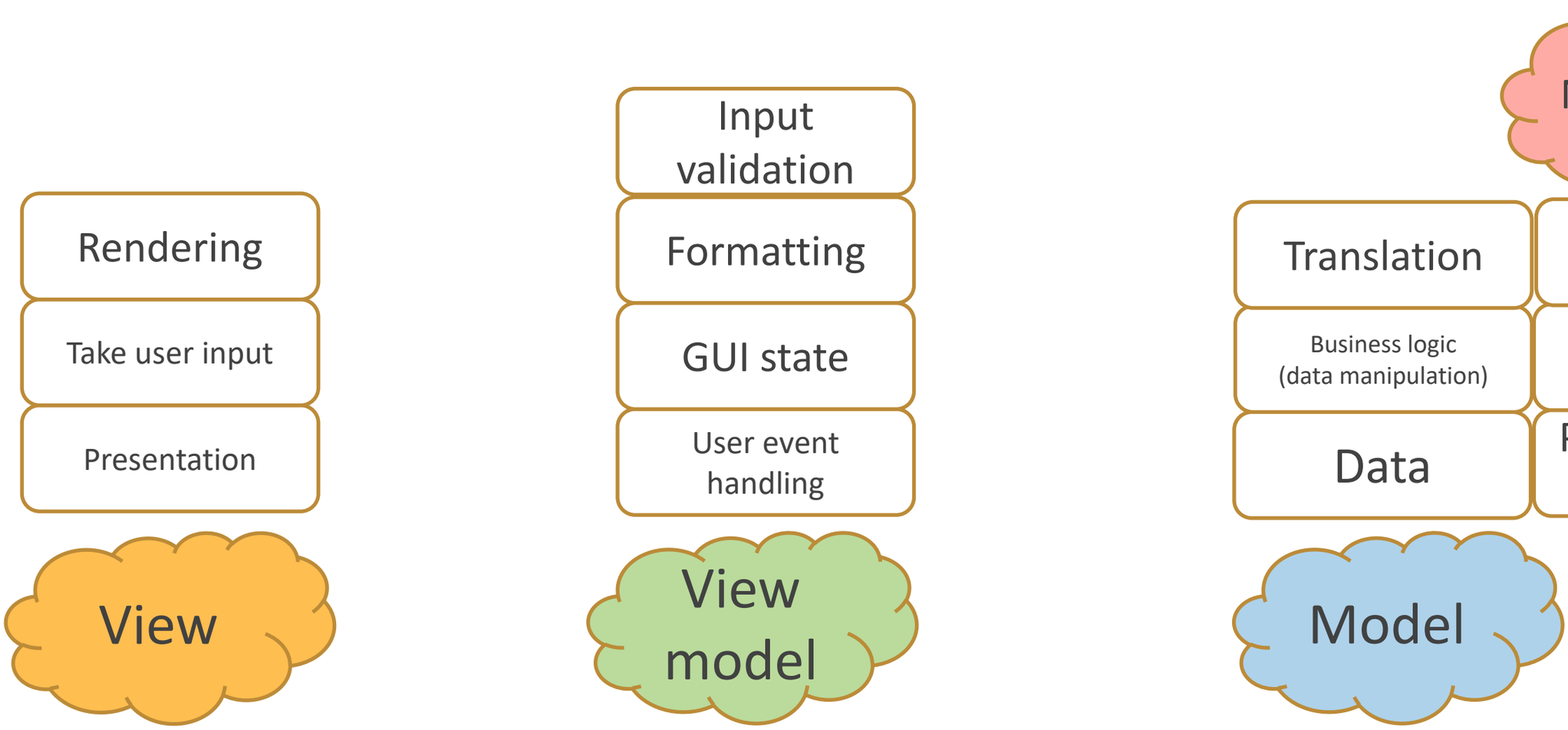
Model, View, View-model (MVVM)
Part 2

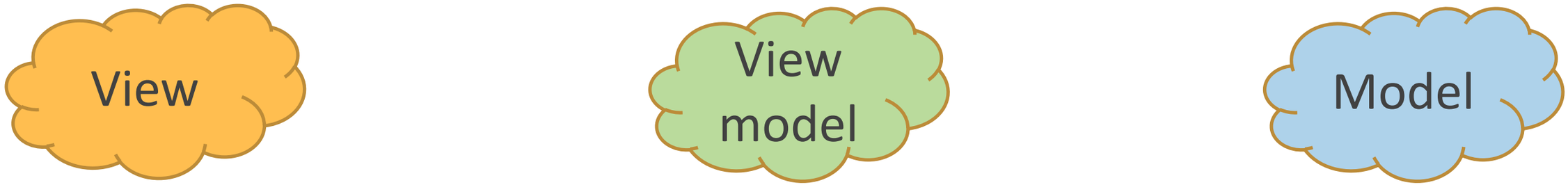
Agenda

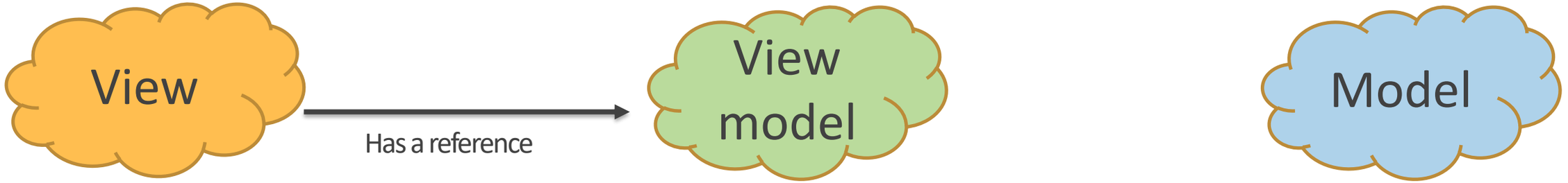
- Communication between the different parts

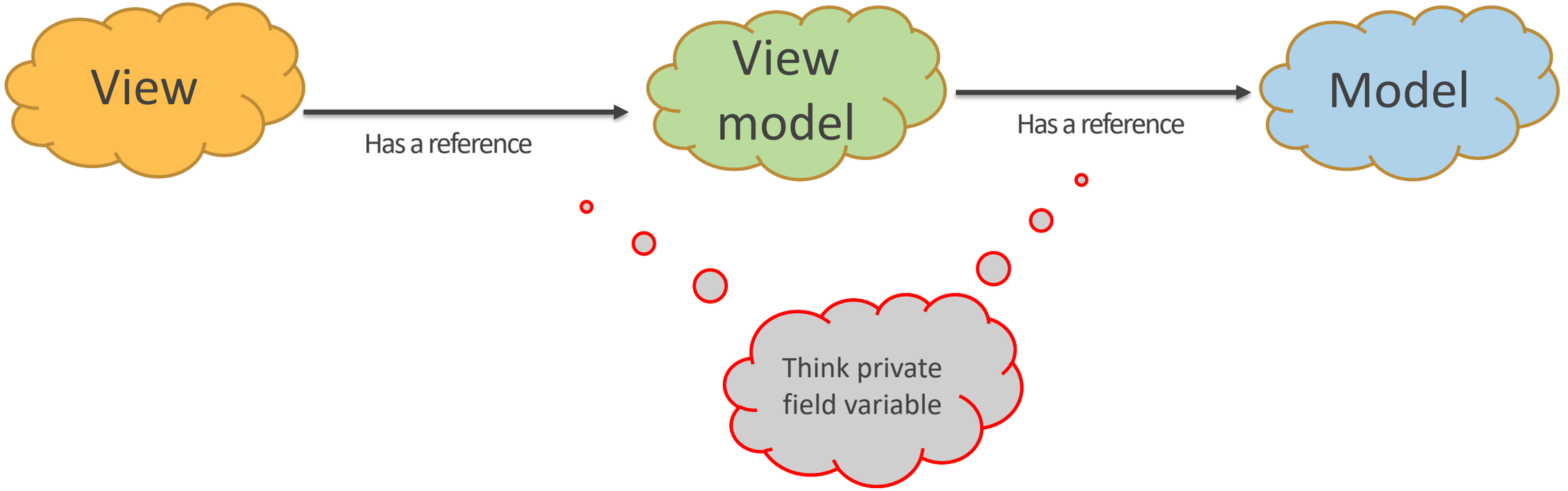
Dependencies

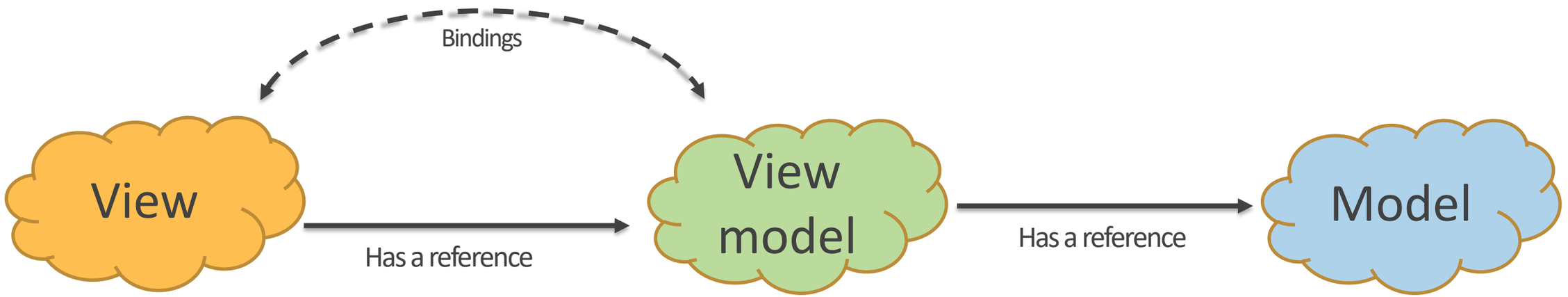
- Who knows about who, and how does communication happen?


Rendering
Take user input
Presentation
View
Input validation
Formatting
GUI state
User event handling
View model
Networking
Translation
Data validation
Business logic (data manipulation)
Storage/ data base
Data
Parsing (XML, JSON)
Model

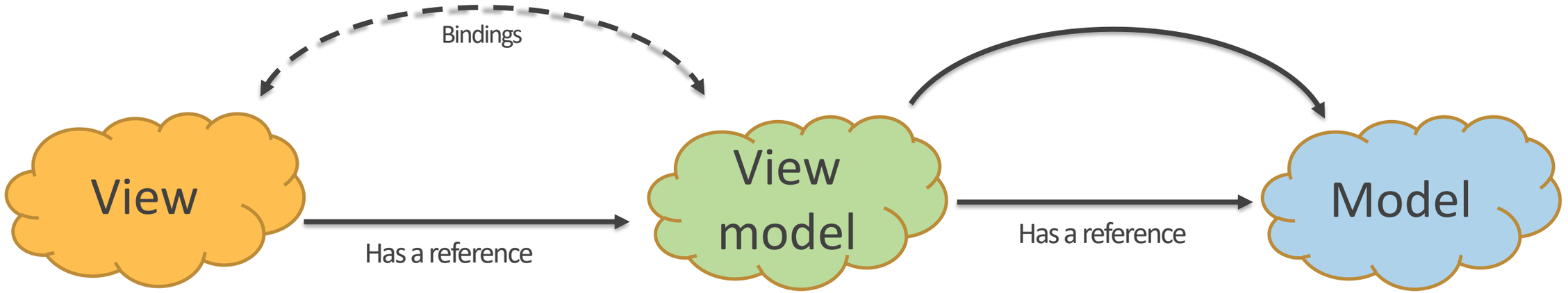

View
View model
Model

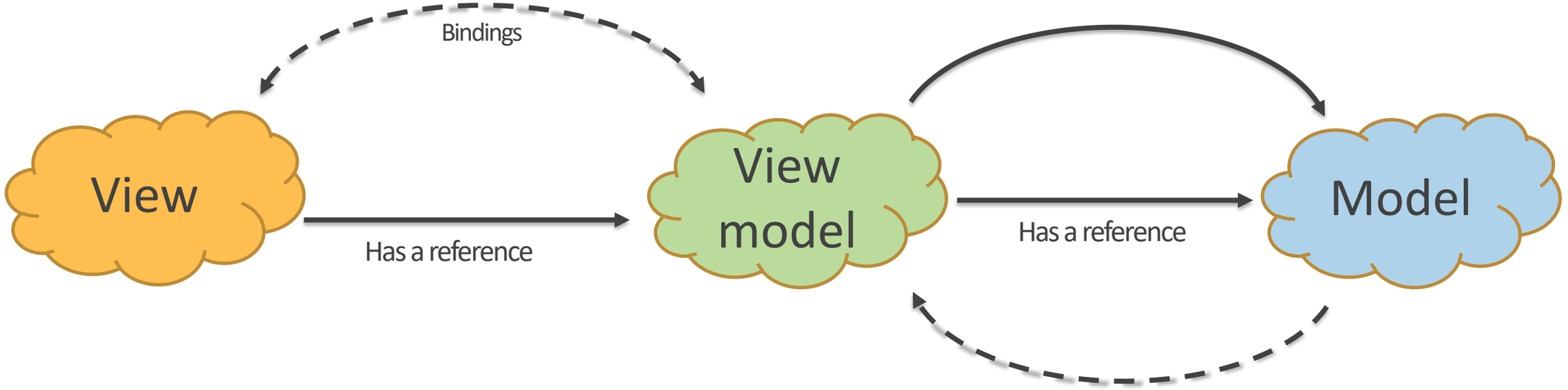

View
Has a reference
View model
Model

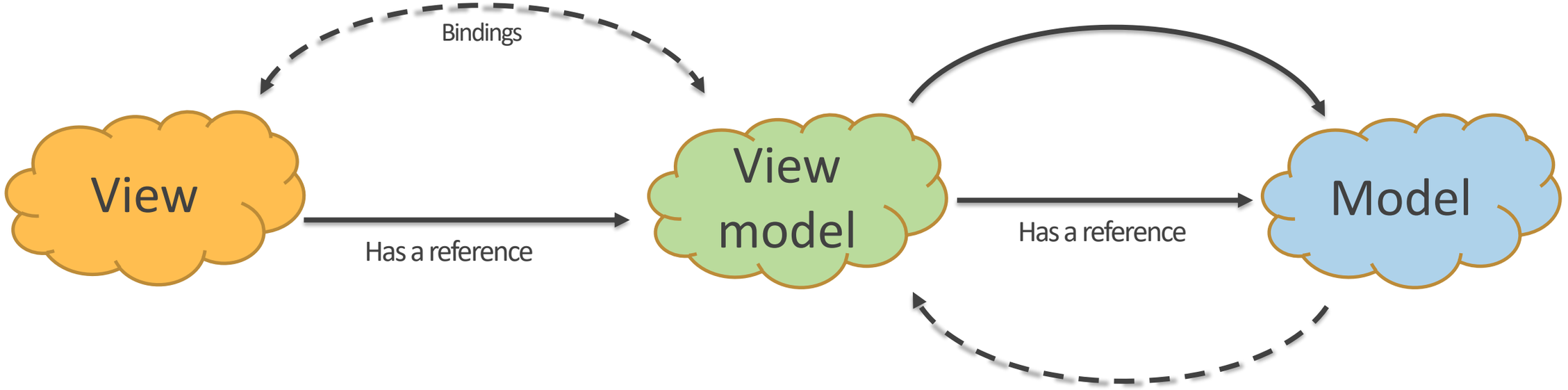

View
Has a reference
View model
Has a reference
Model
Think private field variable


Bindings
View
Has a reference
View model
Has a reference
Model

- The View model calls methods on the model

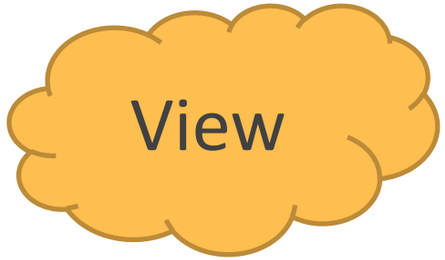


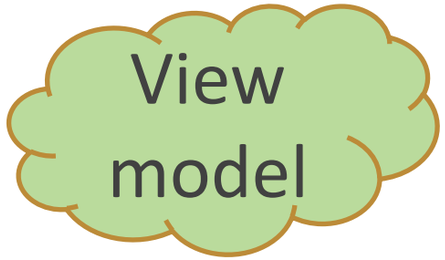

Bindings
View
Has a reference
View model
Has a reference
Model

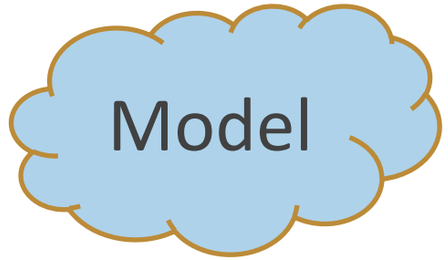

Bindings
View
Has a reference
View model
Has a reference
Model

In JavaFX

- So far a lot of fluffy concepts
- How do we translate this to JavaFX?

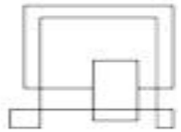






End of part 2

Life is great
VIA University College



Software Development with UML and Java 2

Model, View, View-model (MVVM)
Part 3

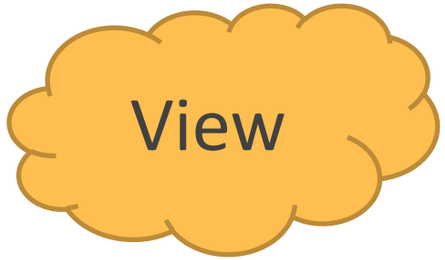
Agenda

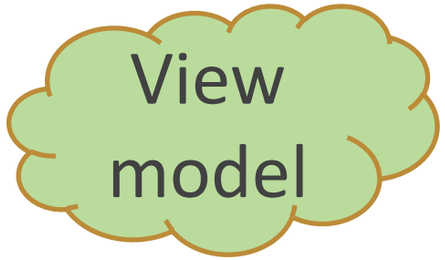
- UML diagrams, how to structure your classes

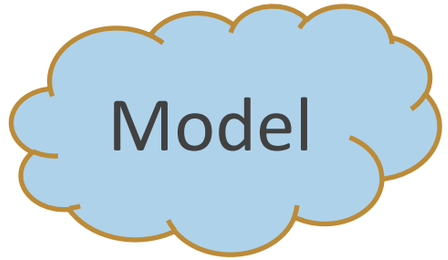
Best practices for JavaFX?

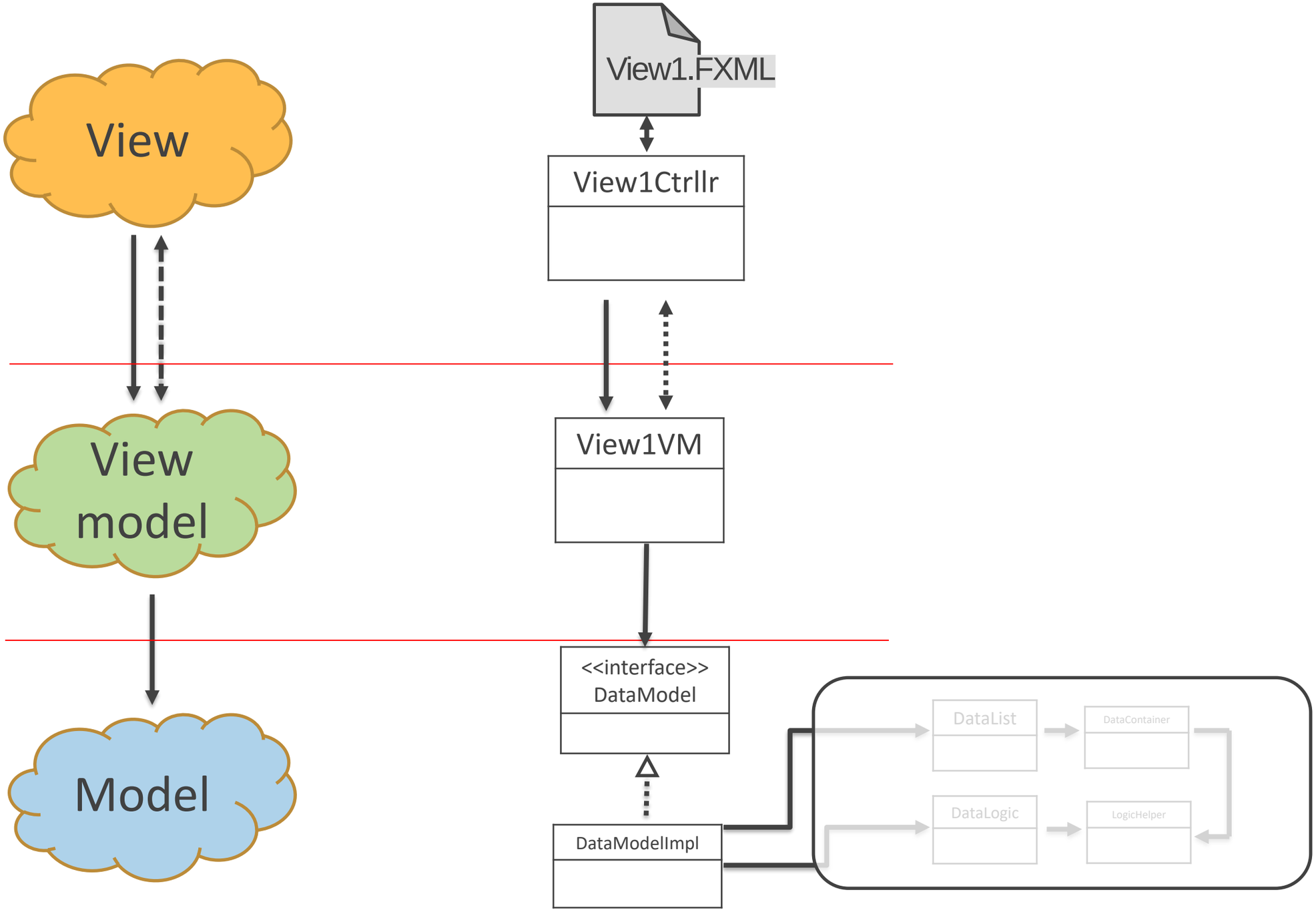
In JavaFX

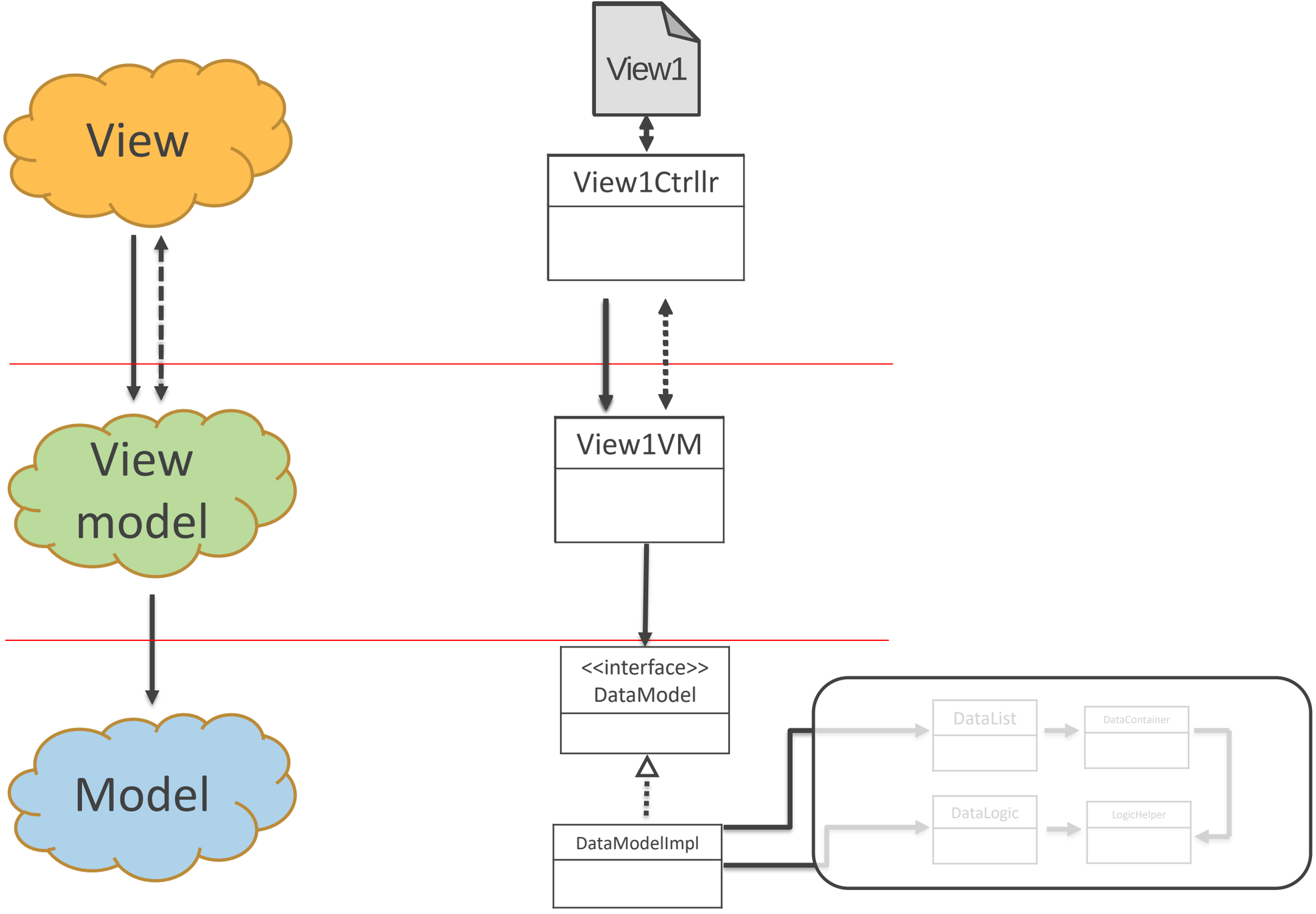
- So far a lot of fluffy concepts
- How do we translate this to JavaFX?

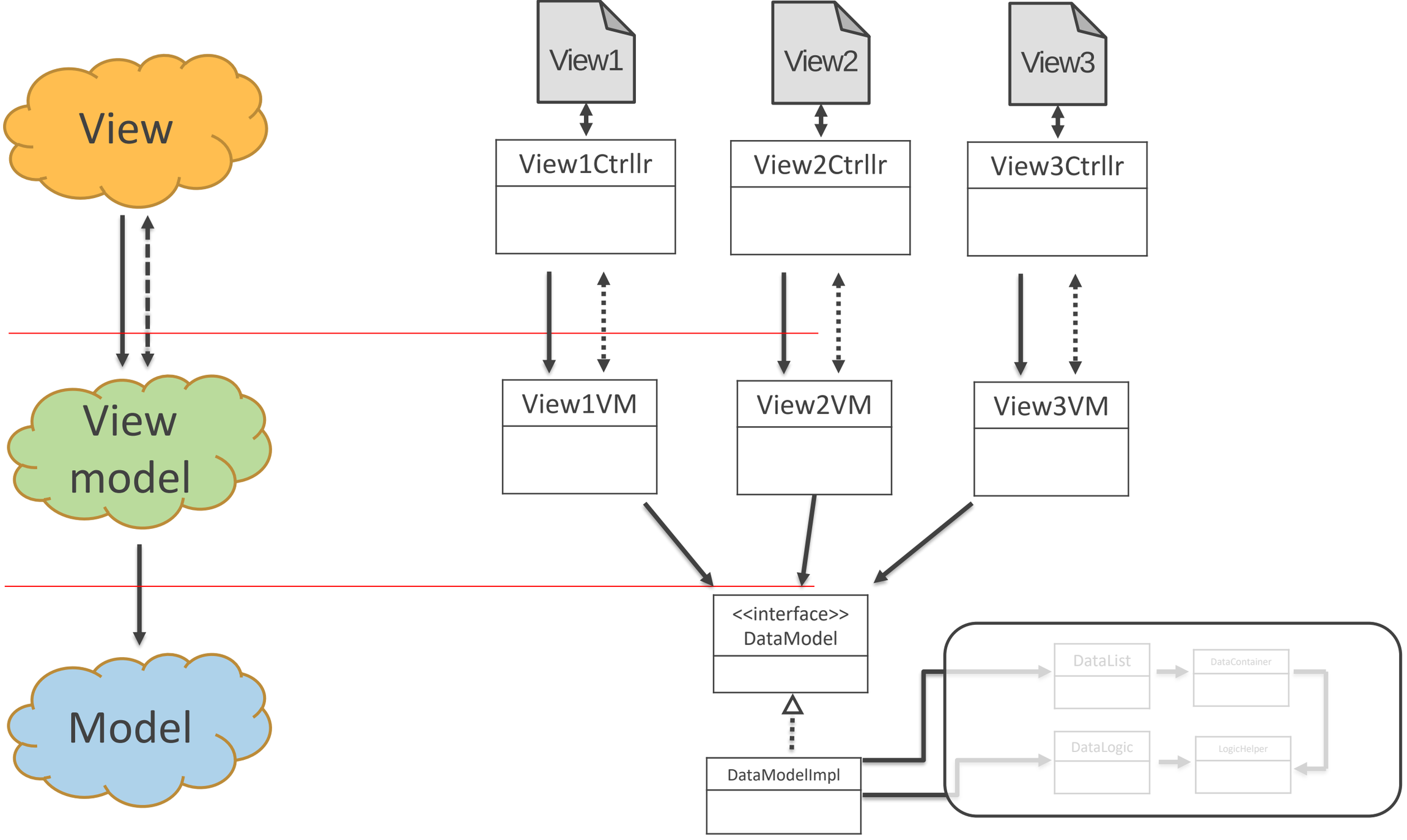


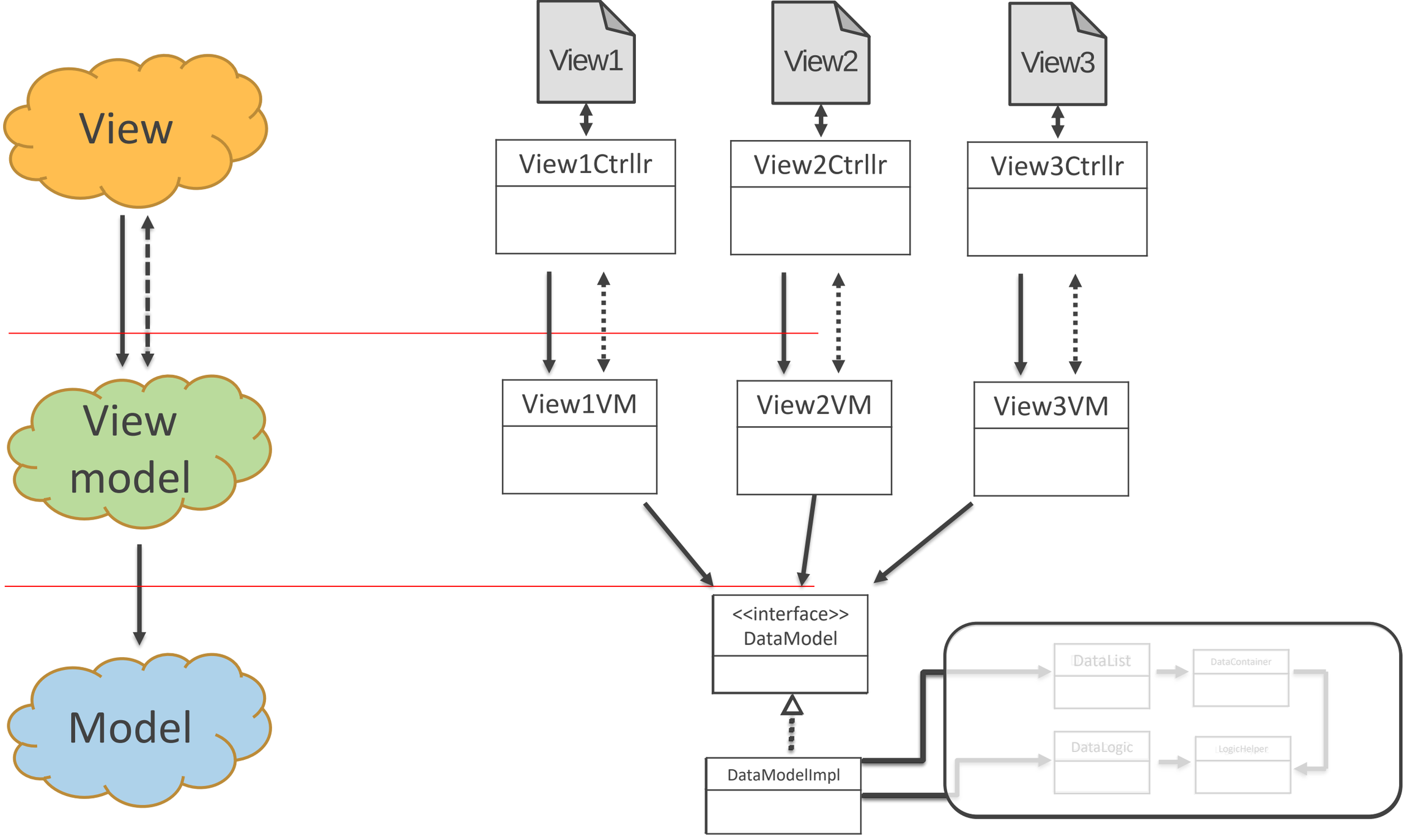


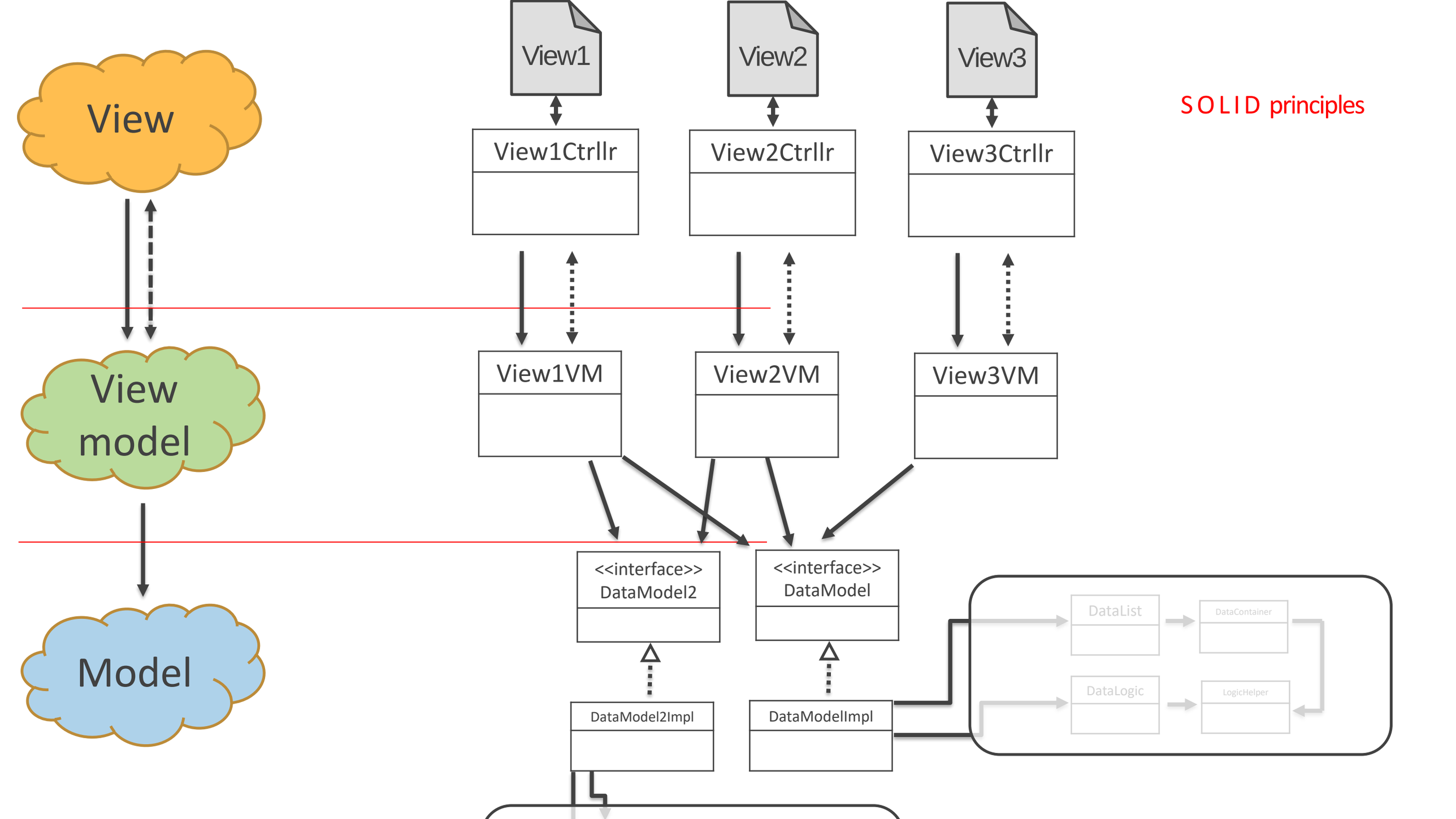


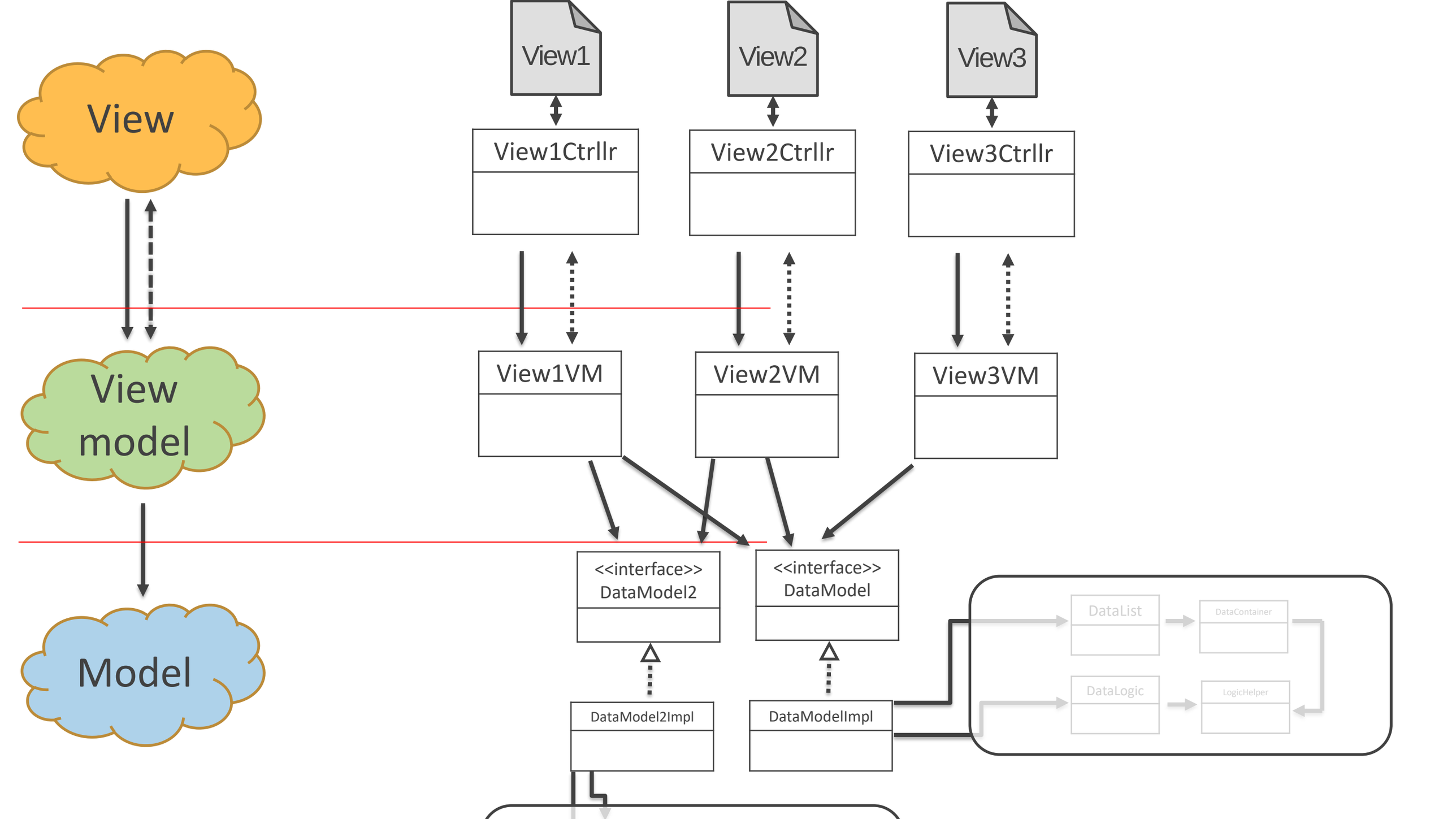

View
View model
Model
View1.FXML
View1Ctrllr
View1VM
<<interface>>
DataModel
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper


View
View model
Model
View1
View1Ctrllr
View1VM
<<interface>>
DataModel
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper


View
View model
Model
View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper

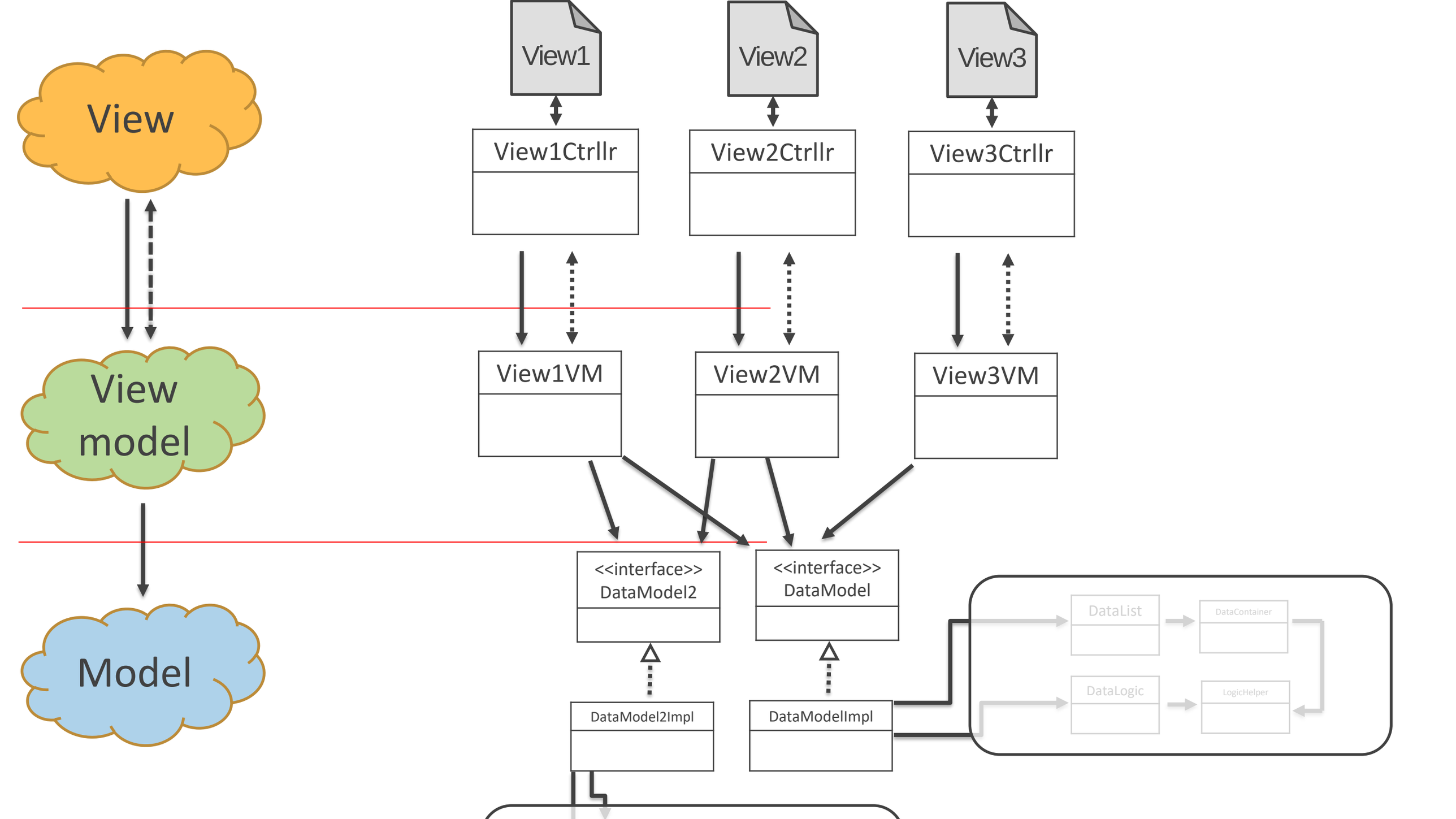

View
View model
Model
View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper

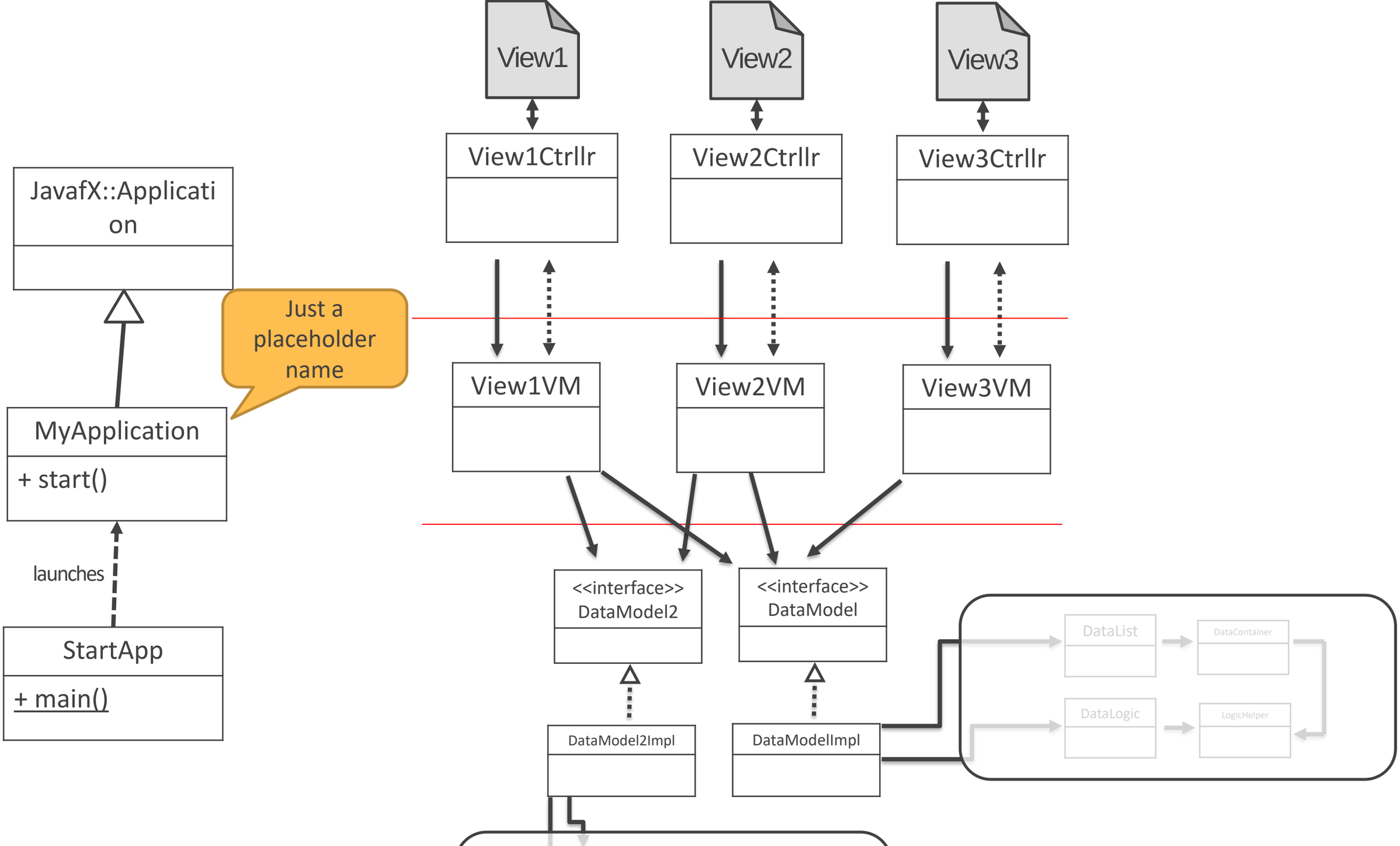

View
View model
Model
View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper
S O L I D principles

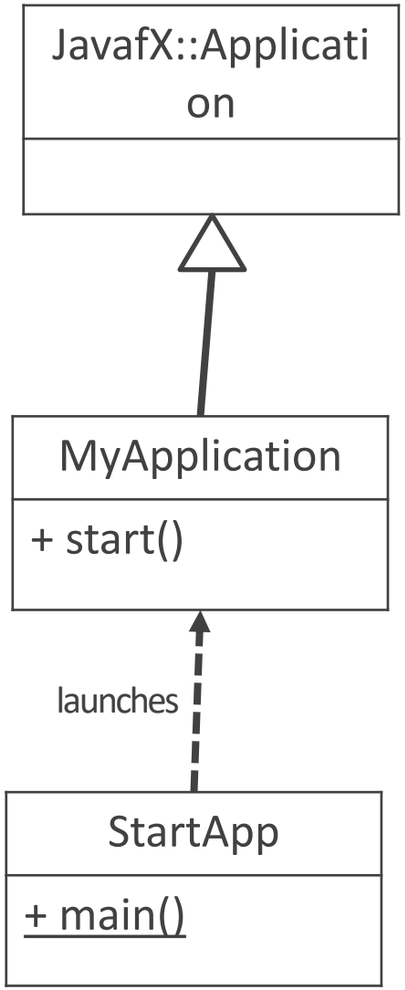

View
View model
Model
View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper

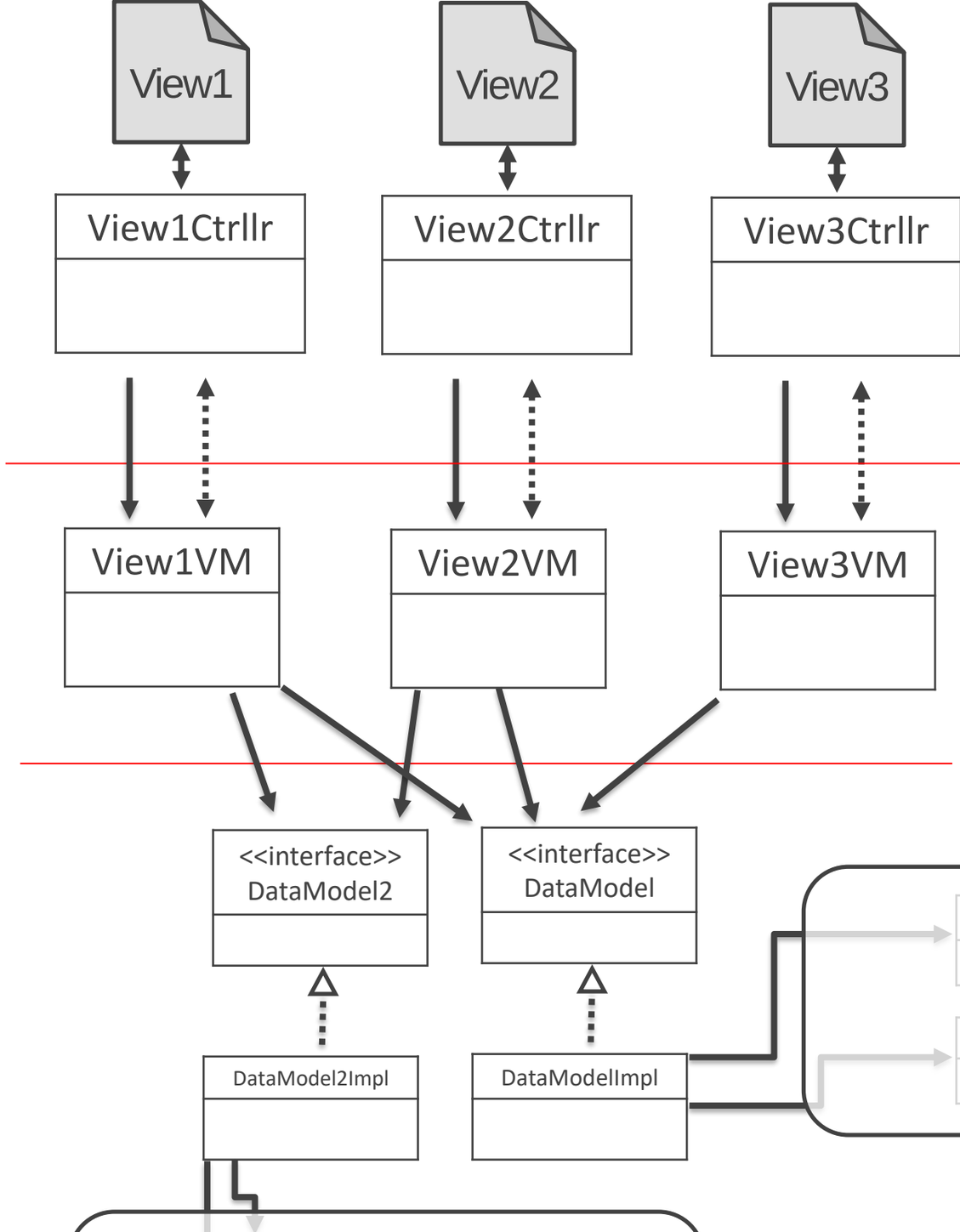
MVVM, bind everything together

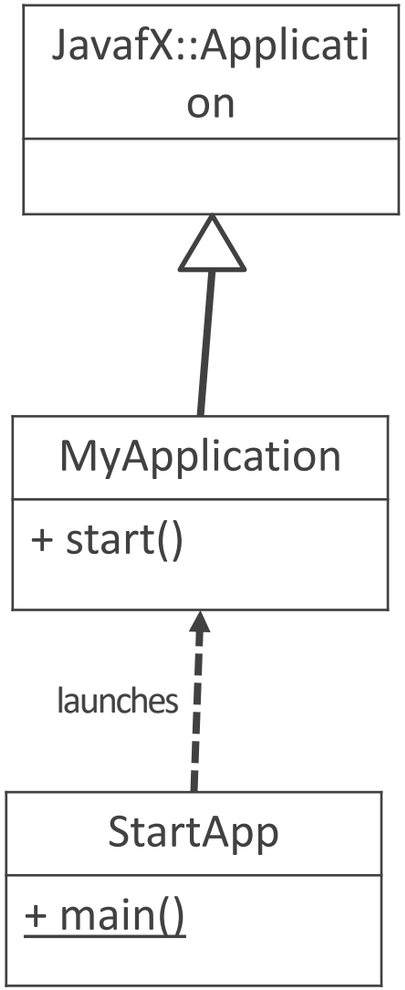
- We have the core structure of our program, but how is everything started? When are the classes created?
- Let’s introduce classes responsible for this.


View
View model
Model
View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper


View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
JavafX::Applicati
on
Just a
placeholder
name
View1VM
View2VM
View3VM
MyApplication
+ start()
launches
StartApp
+ main()
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl
DataList
DataContainer
DataLogic
LogicHelper


JavafX::Application
MyApplication
+ start()
launches
StartApp
+ main()

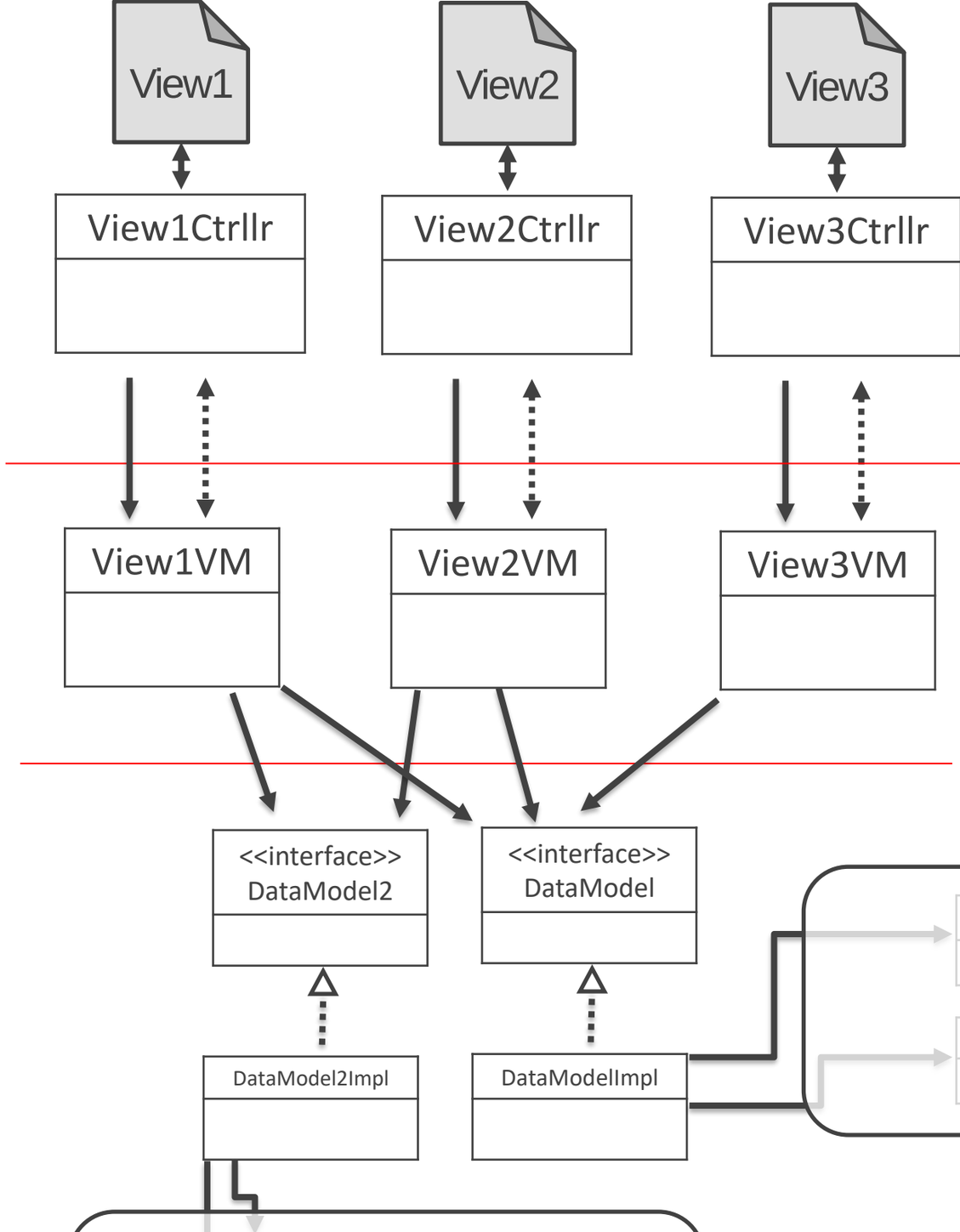

View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl

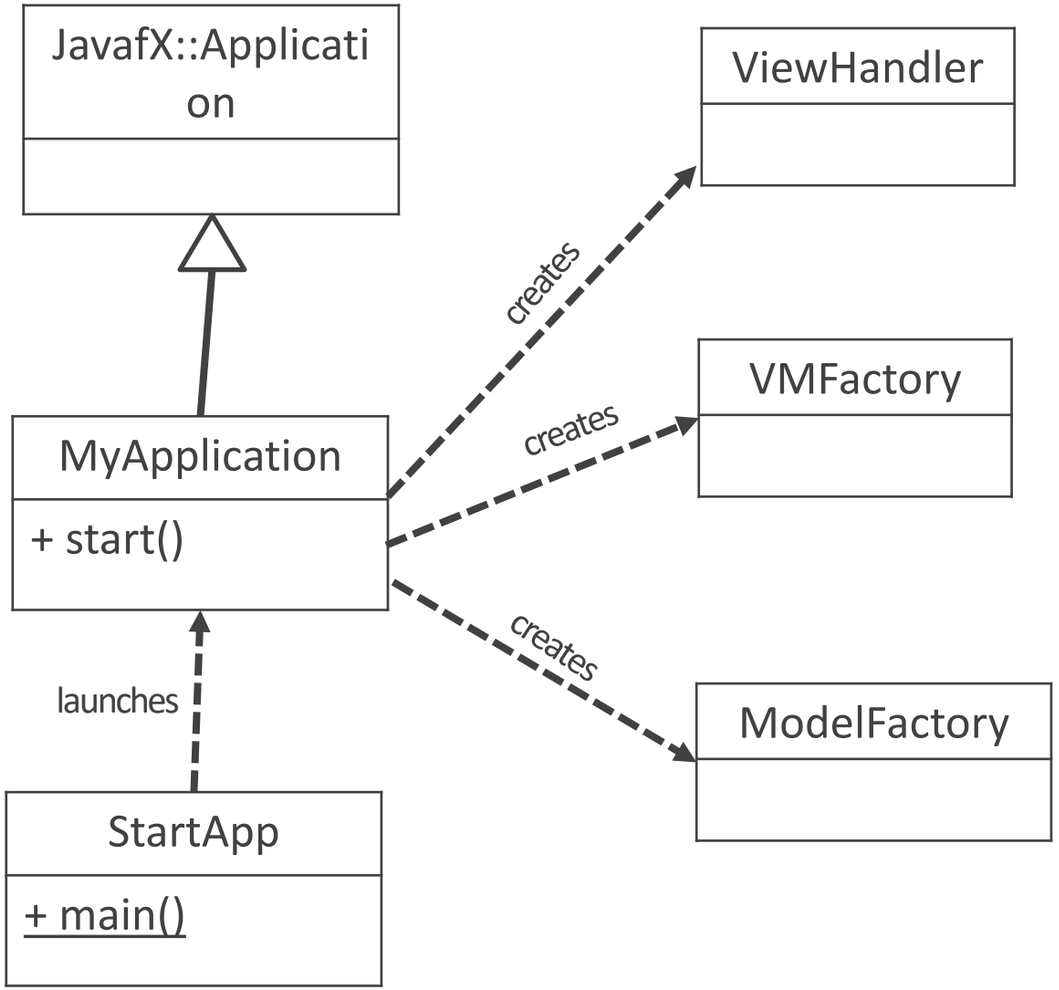


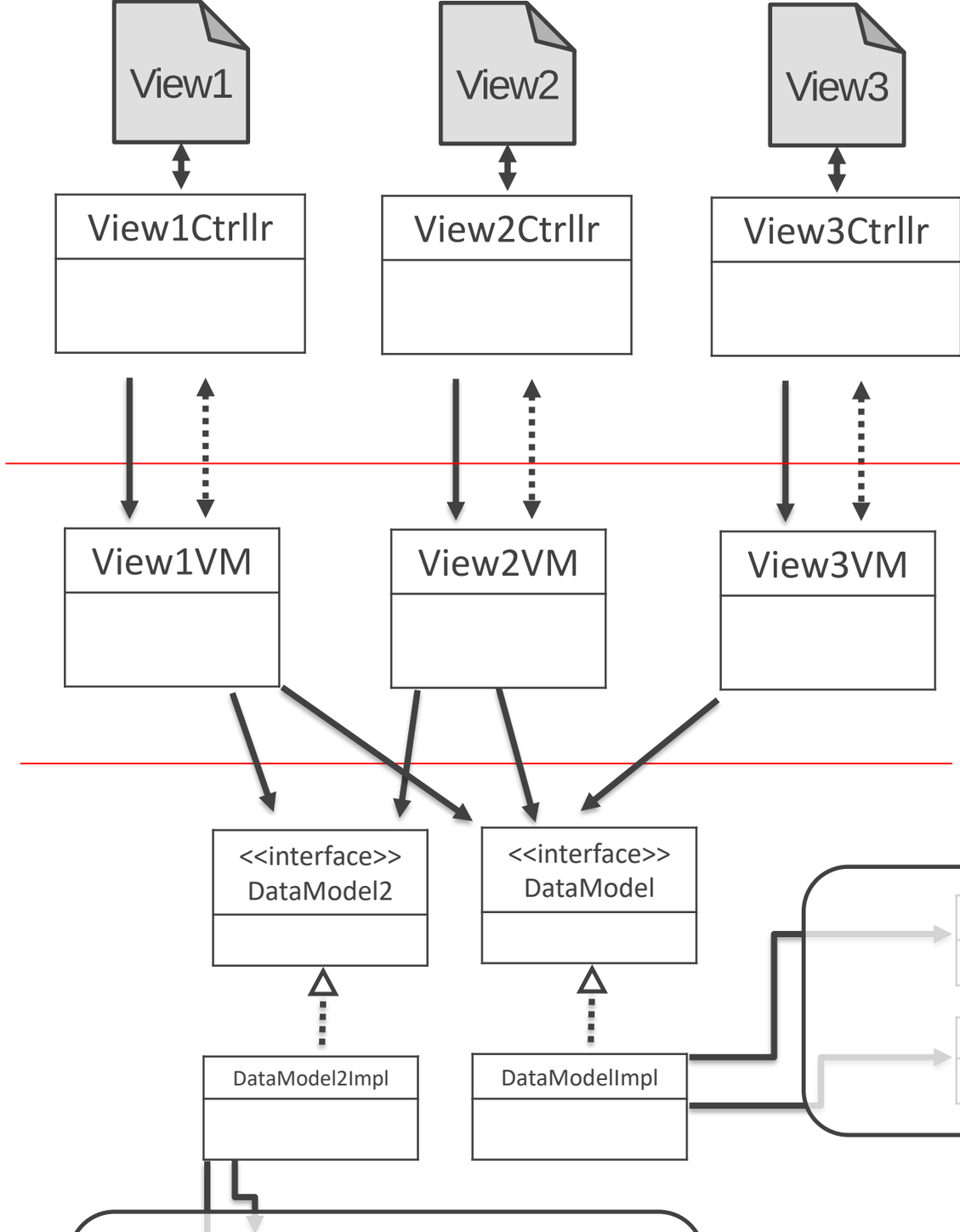
ViewHandler

VMFactory

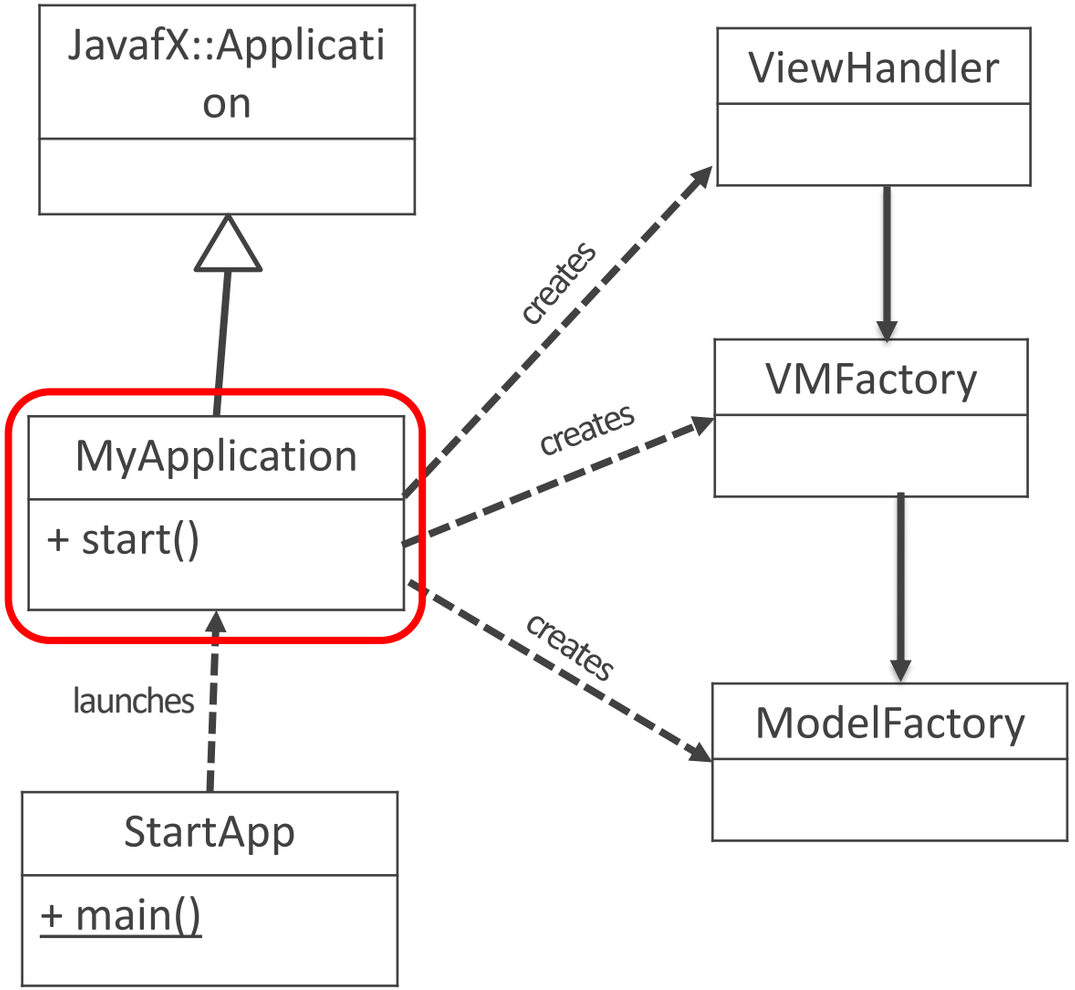
ModelFactory

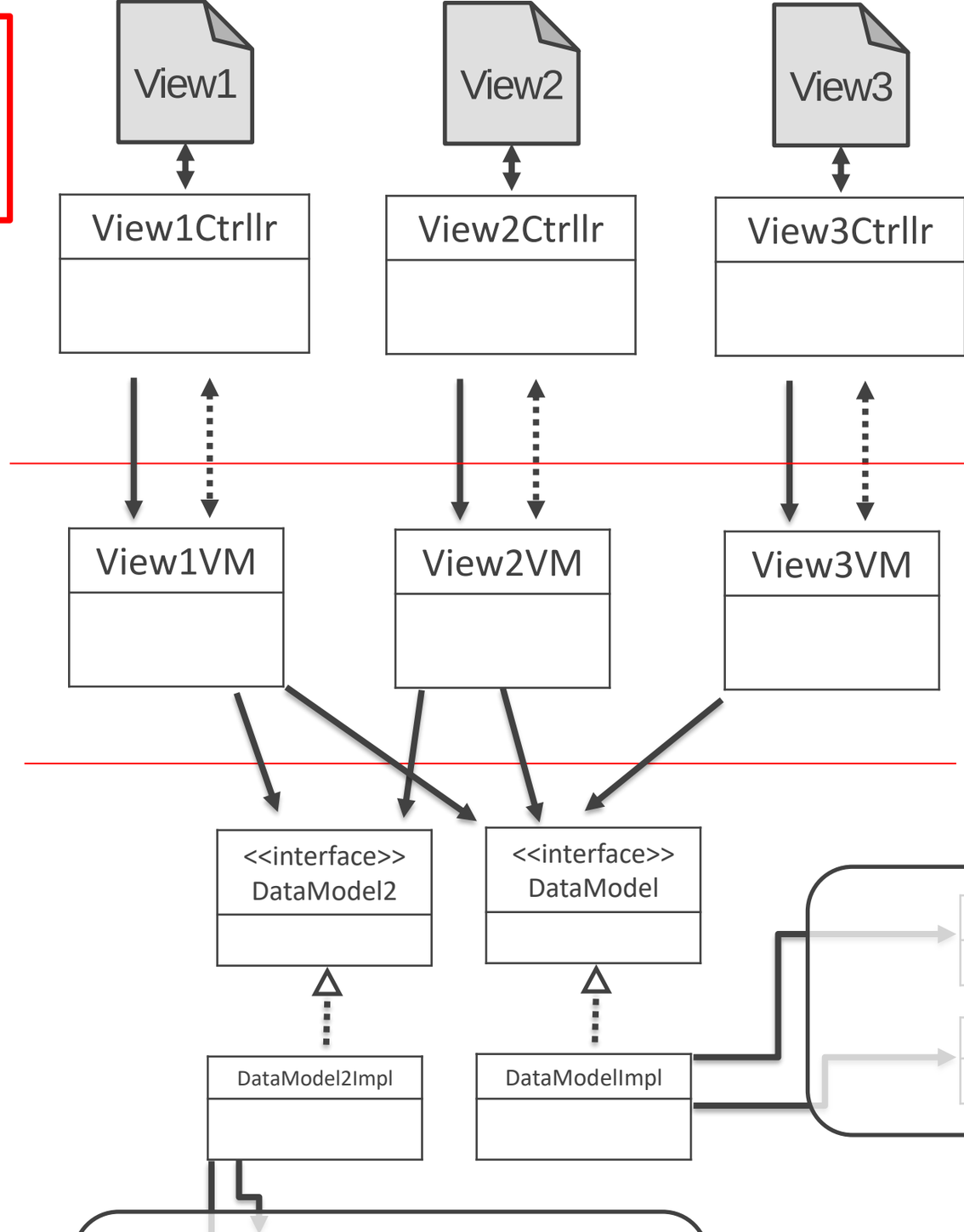


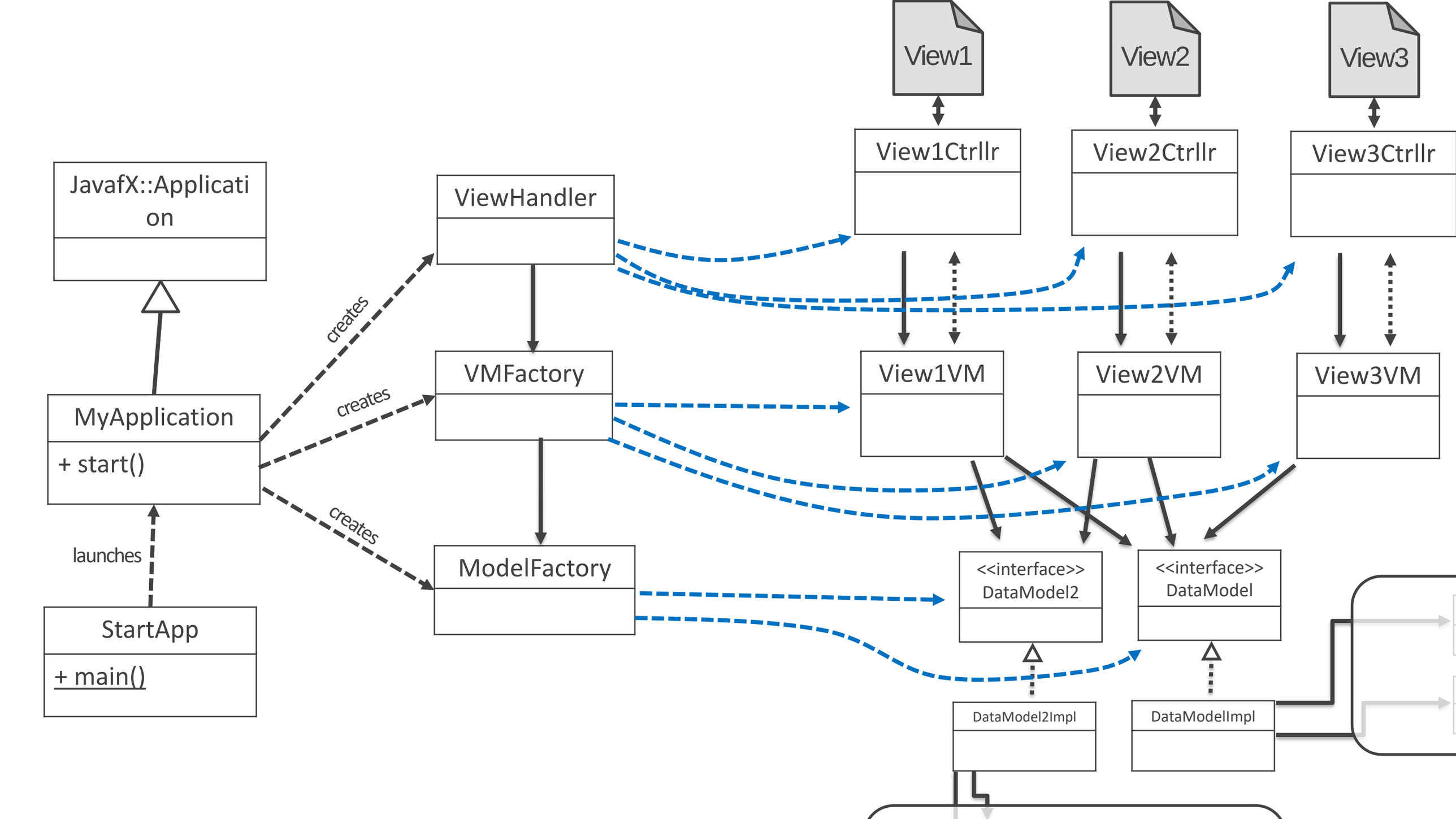

JavafX::Application
ViewHandler
VMFactory
ModelFactory
MyApplication
+ start()
StartApp
+ main()
creates
creates
creates
launches

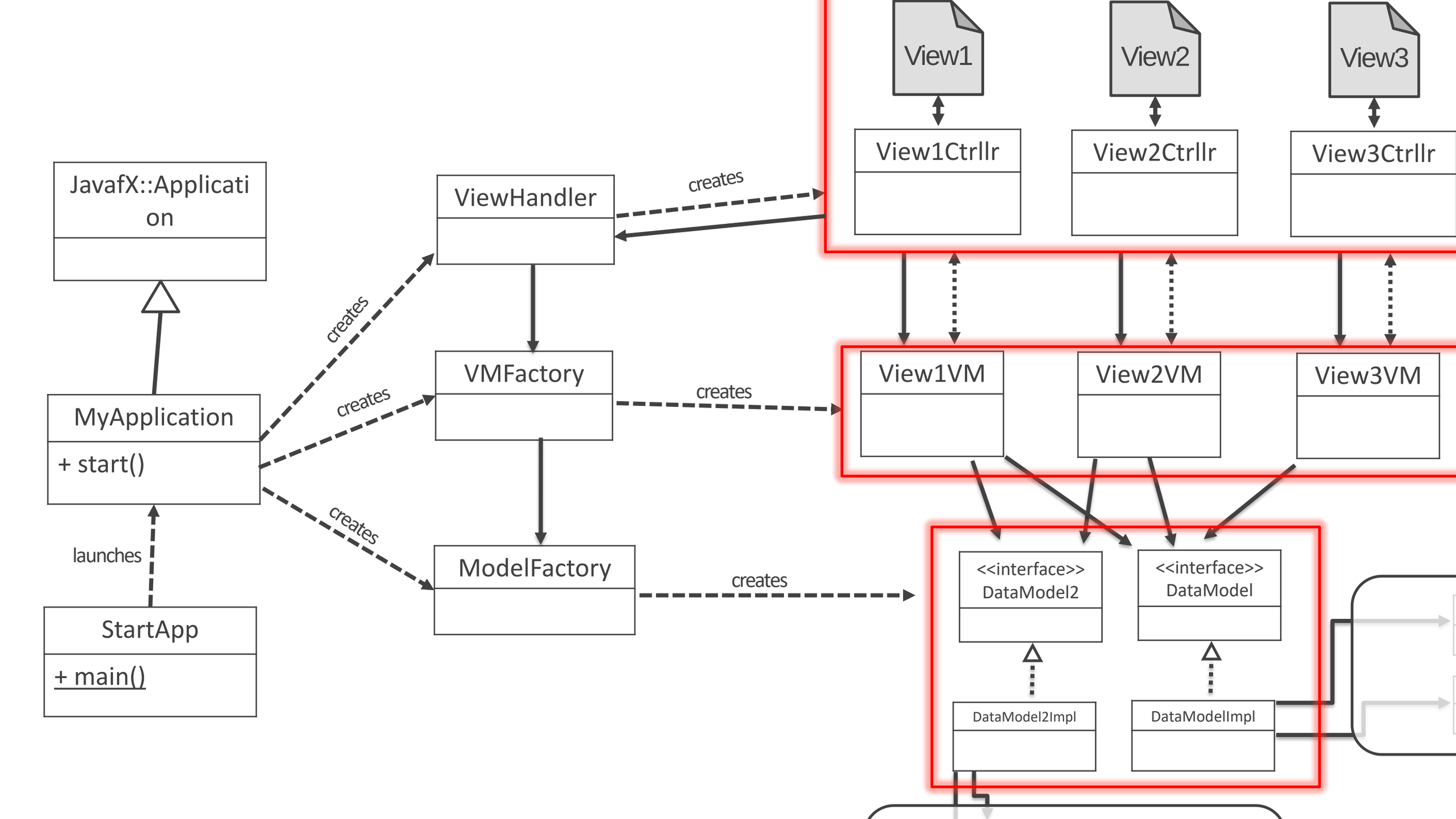

View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl

```
ModelFactory mf = new ModelFactory();
ViewModelFactory vmf = new ViewModelFactory(mf);
ViewHandler vh = new ViewHandler(vmf);
```



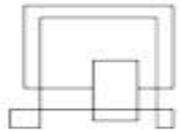



View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
JavafX::Application
ViewHandler
VMFactory
ModelFactory
MyApplication
+ start()
StartApp
+ main()
launches
creates
creates
creates
View1VM
View2VM
View3VM
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl


View1
View2
View3
View1Ctrllr
View2Ctrllr
View3Ctrllr
JavafX::Applicati
on
ViewHandler
creates
creates
creates
creates
VMFactory
creates
View1VM
View2VM
View3VM
MyApplication
+ start()
launches
StartApp
+ main()
ModelFactory
creates
<<interface>>
DataModel2
<<interface>>
DataModel
DataModel2Impl
DataModelImpl

End of part 3

Life is great
VIA University College



Software Development with UML and Java 2

Model, View, View-model (MVVM)

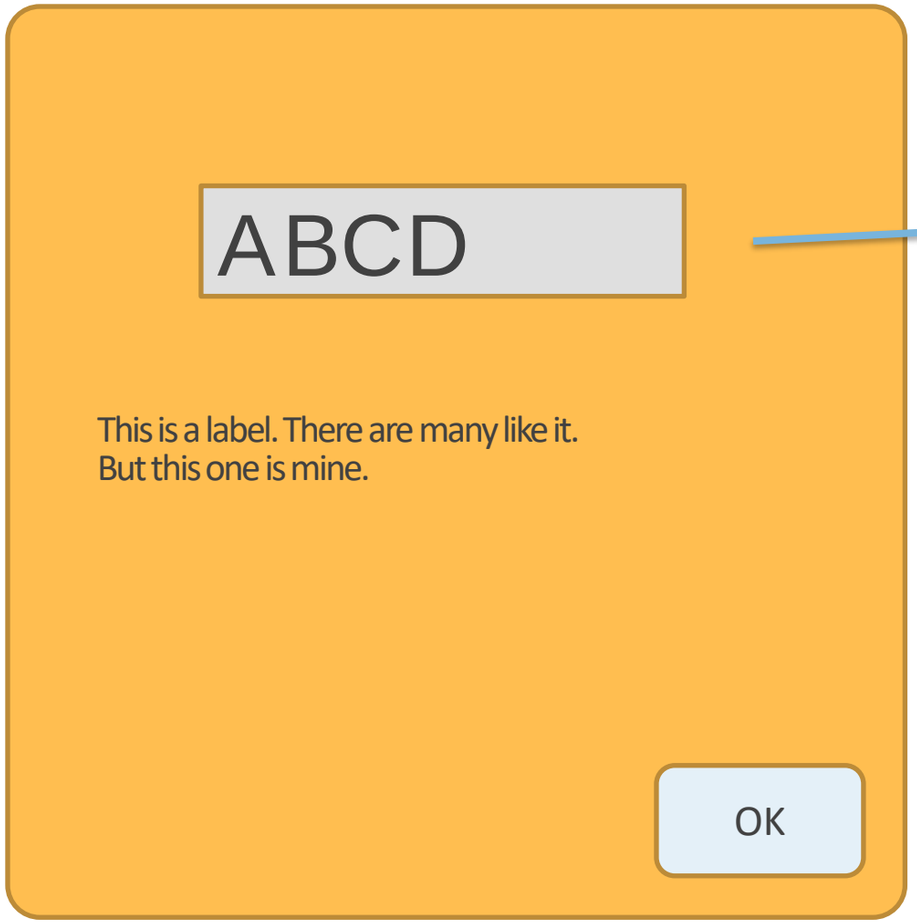
Part 4

Agenda

- Previously:
  1. Concept, motivation, different parts, responsibilities
  2. Communication and interaction
  3. UML Template
- Code and examples

GUI

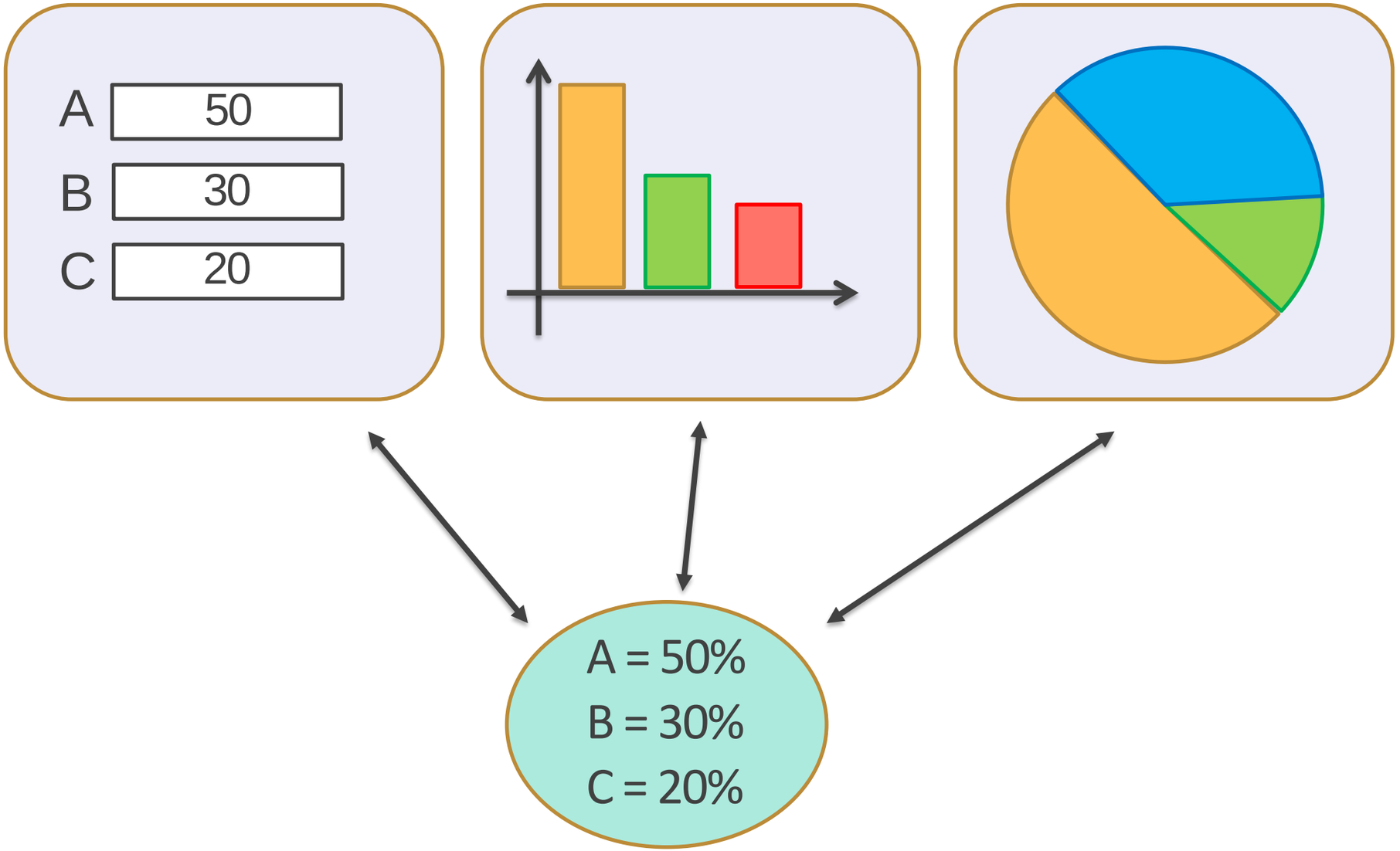
ViewModel

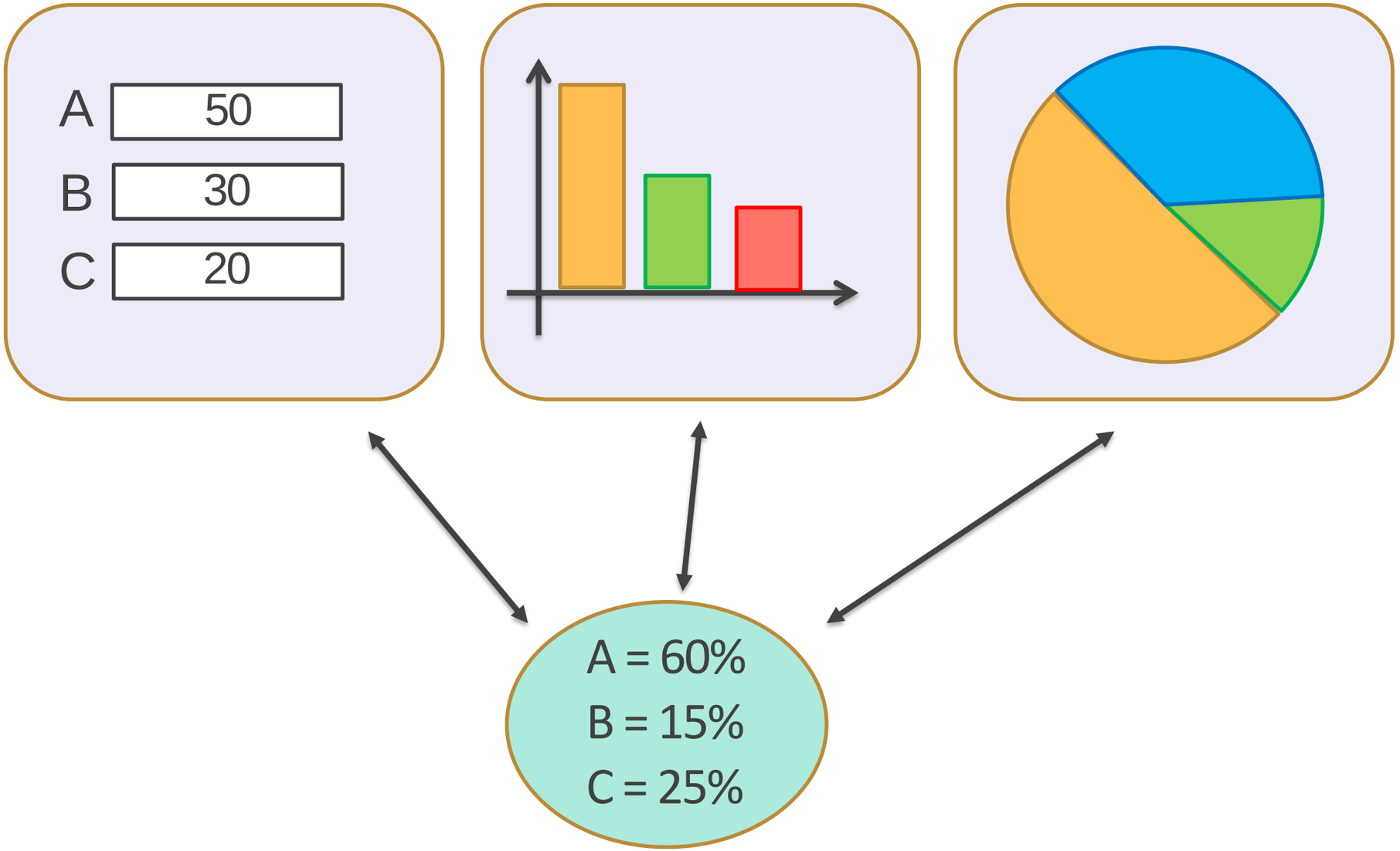


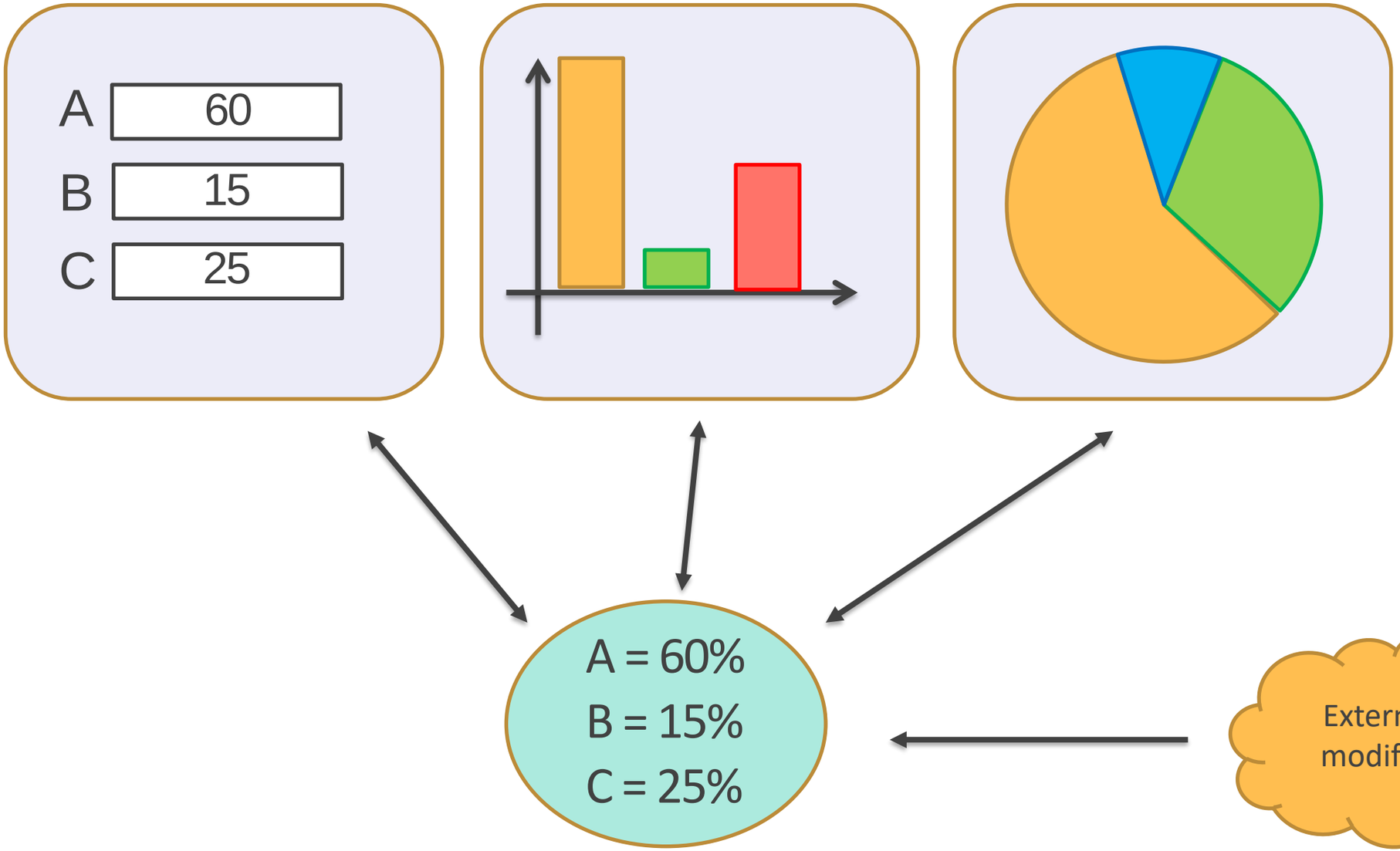
Private StringProperty text;

ABCD

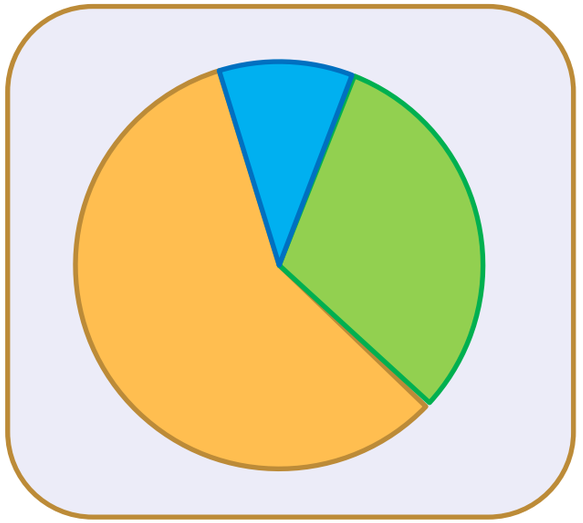
If a Property in the ViewModel is updated, it can be reflected in the GUI

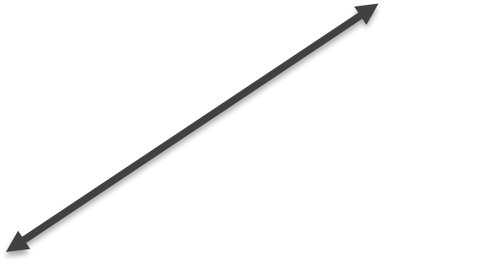

A
50
B
30
C
20
A = 50%
B = 30%
C = 20%


A
50
B
30
C
20
A = 60%
B = 15%
C = 25%


A
60
B
15
C
25
A = 60%
B = 15%
C = 25%
External modifier

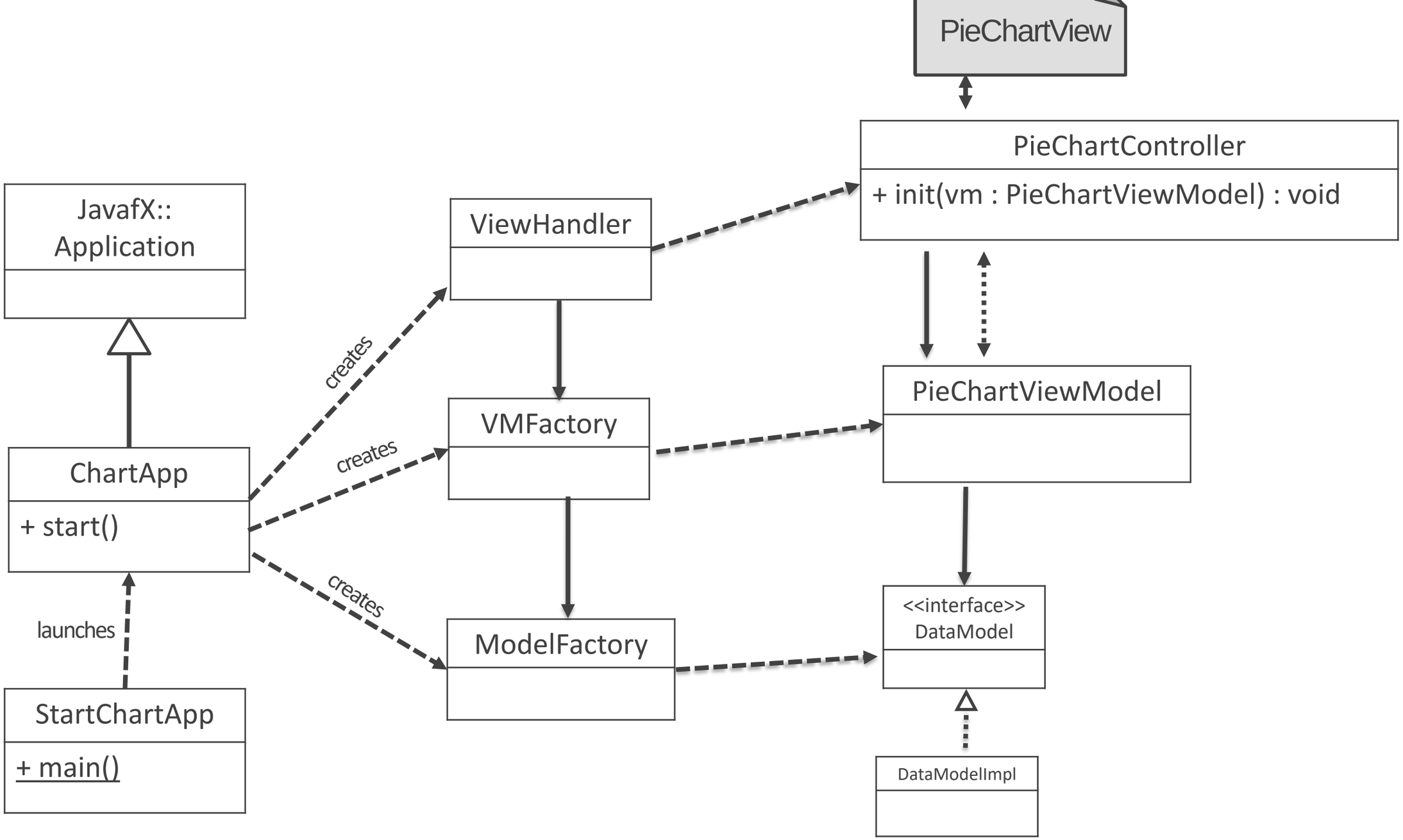
– We’ll start with the Pie Chart view



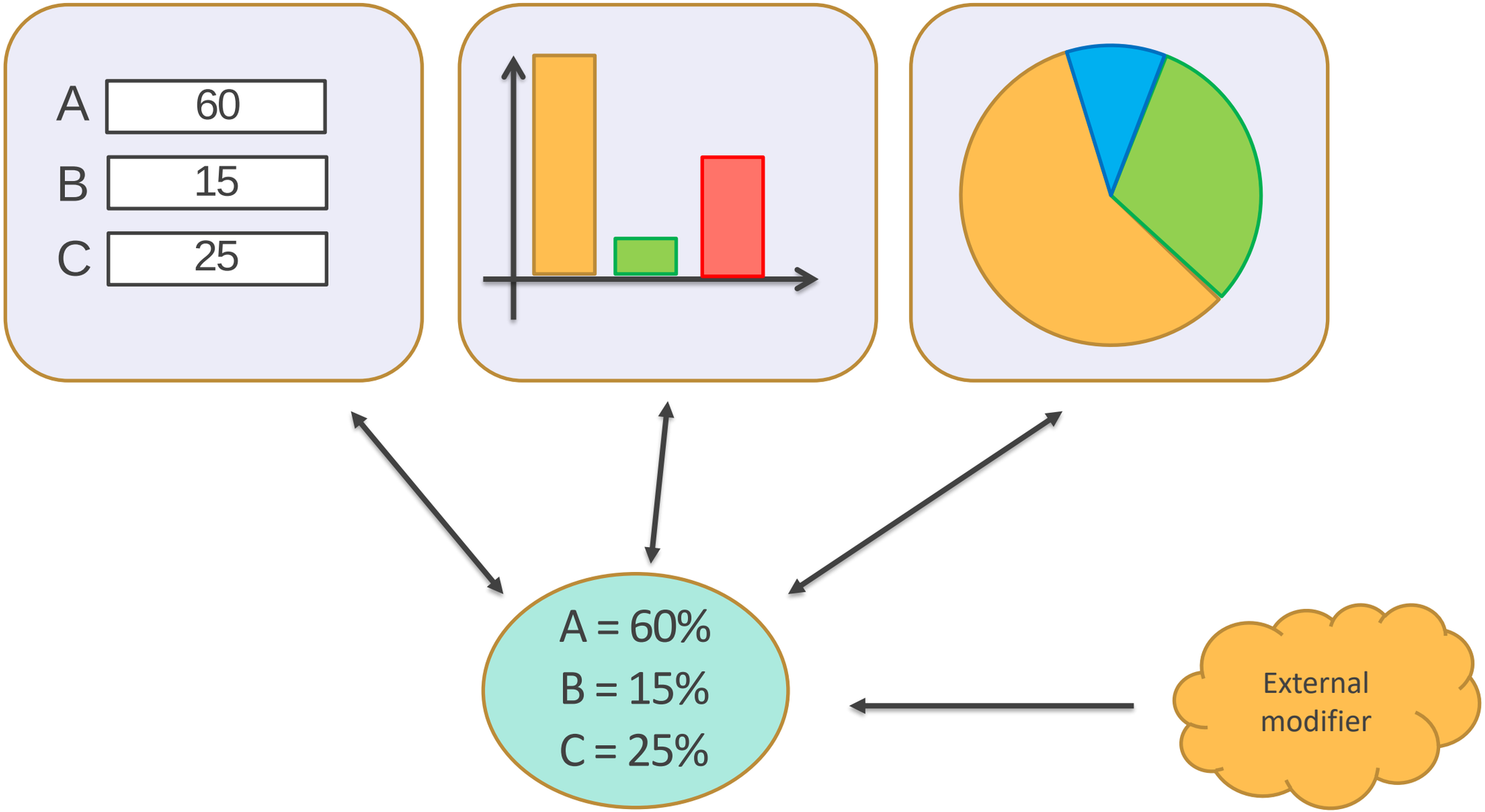


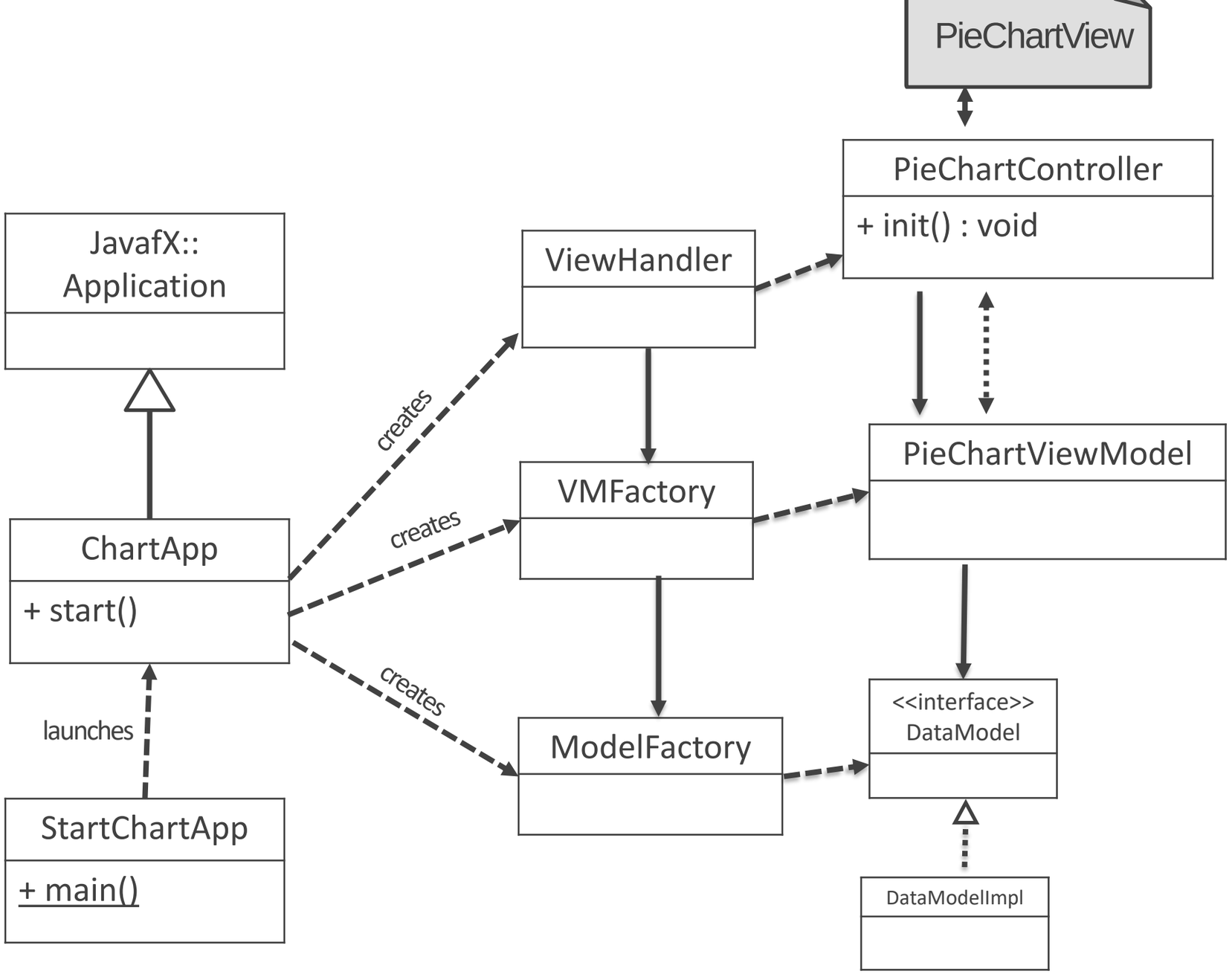
A = 60%
B = 15%
C = 25%

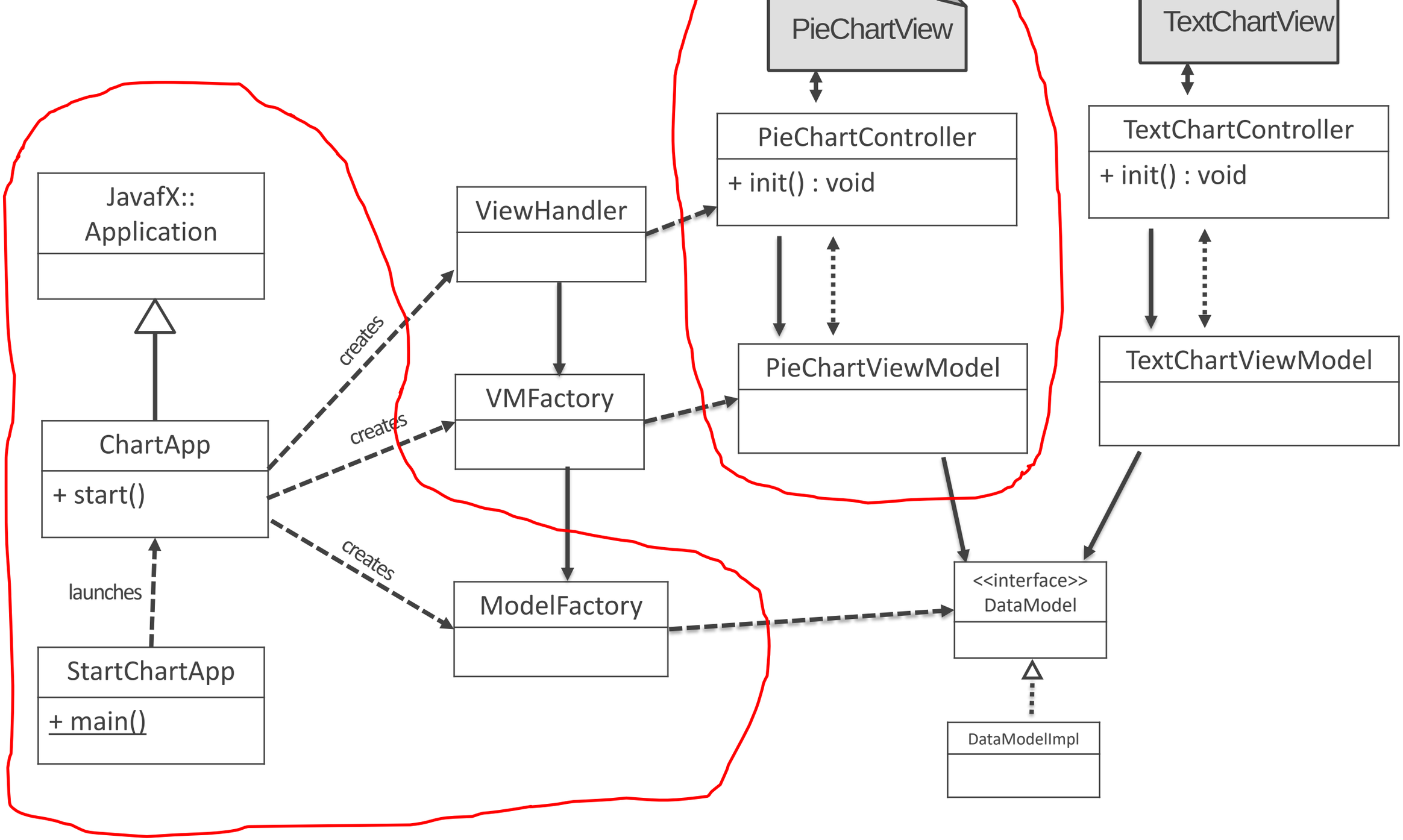
External modifier


PieChartView
PieChartController
+ init(vm : PieChartViewModel) : void
JavafX::
Application
ViewHandler
creates
creates
creates
ChartApp
+ start()
VMFactory
PieChartViewModel
launches
ModelFactory
<<interface>>
DataModel
StartChartApp
+ main()
DataModelImpl

The program




PieChartView
PieChartController
+ init() : void
JavafX::
Application
ViewHandler
creates
creates
creates
VMFactory
PieChartViewModel
ChartApp
+ start()
launches
ModelFactory
<<interface>>
DataModel
StartChartApp
+ main()
DataModelImpl


PieChartView
TextChartView
PieChartController
+ init() : void
TextChartController
+ init() : void
JavafX::
Application
ViewHandler
VMFactory
ModelFactory
PieChartViewModel
TextChartViewModel
ChartApp
+ start()
StartChartApp
+ main()
launches
creates
creates
creates
<<interface>>
DataModel
DataModelImpl

Bindings again

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel vi

    private PieChart.Data x = ne
    private PieChart.Data y = ne
    private PieChart.Data z = ne

    public PieChartView() {
    }

    public void init(PieChartViewMod                    ChartViewModel) {
        this.viewModel = pieChartV                    Model;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampP

        ObservableList<PieChart.Data> datas = FXCollections.observ
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Bind is one direction.
Changes made in PieChartView::x will be pushed to PieChartViewModel::x.
But not vice versa

If you use the method bindBiDirectional, changes are pushed both was.

Data in the ViewModel is reflected in the View

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

                                                    {
                                                DataValues();



                                                "Last updated: " + model.getLastUpdateTim

                                                erty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Data in the ViewModel is reflected in the View

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTim
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```


```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTim
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTim
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTimeStamp
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class DataModel implements Model {

    private double x;
    private double y;
    private double z;
    private String lastUpdate;

    private Random random = new Random();

    @Override
    public double[] getDataValues() {
        return new double[]{x, y, z};
    }

    @Override
    public String getLastUpdateTimeStamp() {
        return lastUpdate;
    }

    public void recalculateData() {
        int first = random.nextInt(100)+1;
        int second = random.nextInt(100)+1;
        int bottom = Math.min(first, second);
        int top = Math.max(first, second);

        x = bottom;
        y = top - bottom;
        z = 100 - top;
        calTimeStamp();
    }
```

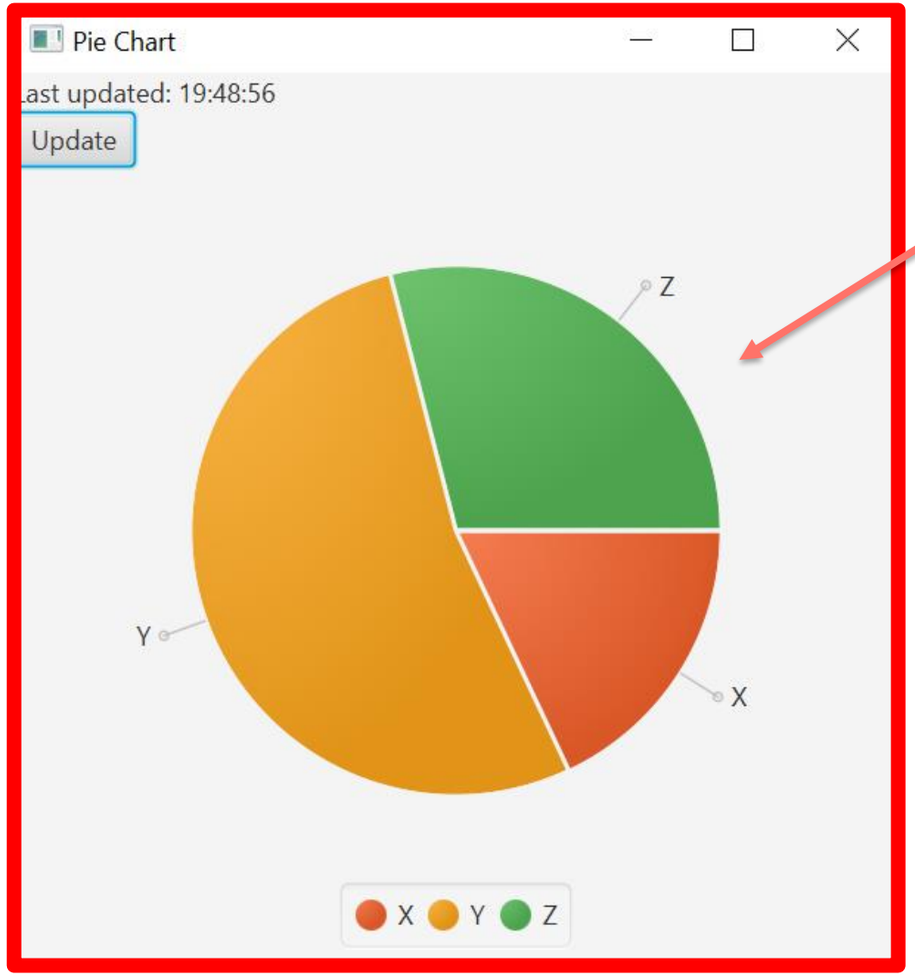
```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTimeStamp
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class DataModel implements Model {

    private double x;
    private double y;
    private double z;
    private String lastUpdate;

    private Random random = new Random();

    @Override
    public double[] getDataValues() {
        return new double[]{x, y, z};
    }

    @Override
    public String getLastUpdateTimeStamp() {
        return lastUpdate;
    }

    public void recalculateData() {
        int first = random.nextInt(100)+1;
        int second = random.nextInt(100)+1;
        int bottom = Math.min(first, second);
        int top = Math.max(first, second);

        x = bottom;
        y = top - bottom;
        z = 100 - top;
        calTimeStamp();
    }
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTim
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```




```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieC

    private Doubl
    private Doubl
    private Doubl
    private Strin

    private Model

    public PieCha
        this.mode
        x = new S
        y = new S
        z = new S
        updateTim
    }

    public void u
        double[]
        x.setValu
        y.setValu
        z.setValu
        updateTim
    }

    public Double
        return x;
    }

    public Double
        return y;
    }

    public Double
        return z;
    }

    public String
        return up
    }
}
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTi
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTi
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTimeStamp
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class DataModel implements Model {

    private double x;
    private double y;
    private double z;
    private String lastUpdate;

    private Random random = new Random();

    @Override
    public double[] getDataValues() {
        return new double[]{x, y, z};
    }

    @Override
    public String getLastUpdateTimeStamp() {
        return lastUpdate;
    }

    public void recalculateData() {
        int first = random.nextInt(100)+1;
        int second = random.nextInt(100)+1;
        int bottom = Math.min(first, second);
        int top = Math.max(first, second);

        x = bottom;
        y = top - bottom;
        z = 100 - top;
        calTimeStamp();
    }
```

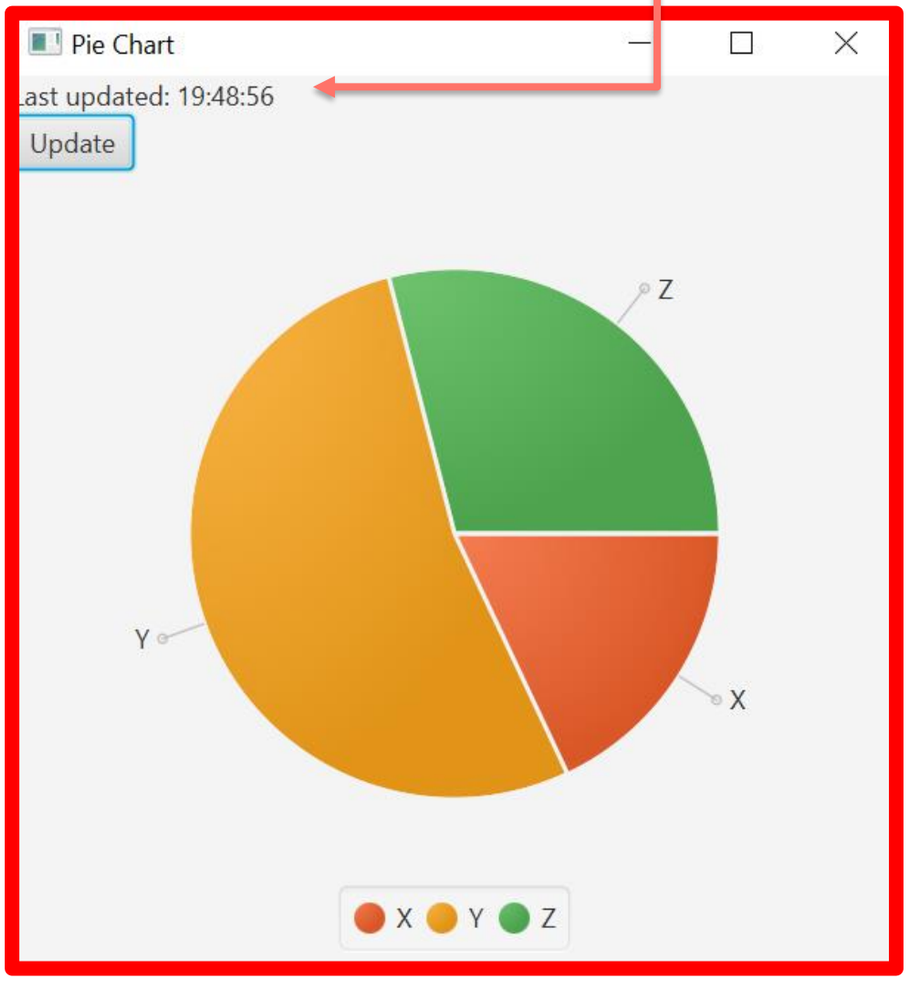
```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTimeStamp
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```

```java
public class DataModel implements Model {

    private double x;
    private double y;
    private double z;
    private String lastUpdate;

    private Random random = new Random();

    @Override
    public double[] getDataValues() {
        return new double[]{x, y, z};
    }

    @Override
    public String getLastUpdateTimeStamp() {
        return lastUpdate;
    }

    public void recalculateData() {
        int first = random.nextInt(100)+1;
        int second = random.nextInt(100)+1;
        int bottom = Math.min(first, second);
        int top = Math.max(first, second);

        x = bottom;
        y = top - bottom;
        z = 100 - top;
        calTimeStamp();
    }
```

```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

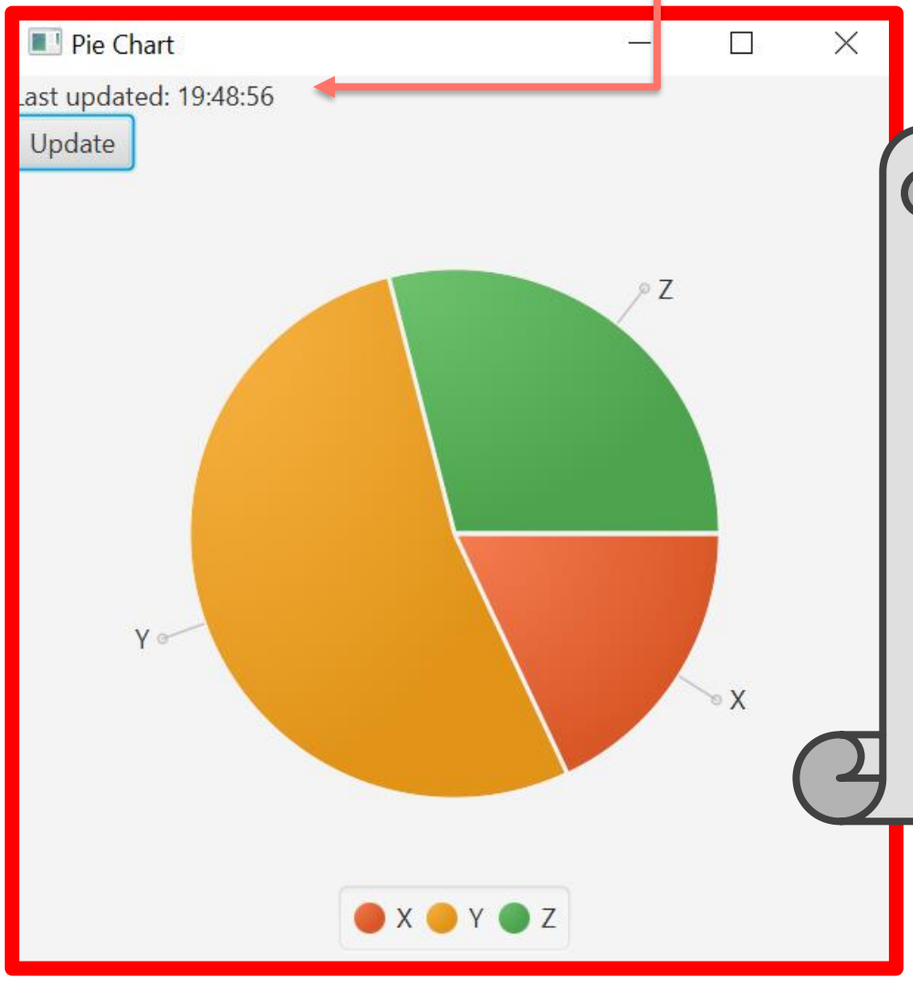
```java
public class PieChartViewModel {

    private DoubleProperty x;
    private DoubleProperty y;
    private DoubleProperty z;
    private StringProperty updateTimeStamp;

    private Model model;

    public PieChartViewModel(Model model) {
        this.model = model;
        x = new SimpleDoubleProperty();
        y = new SimpleDoubleProperty();
        z = new SimpleDoubleProperty();
        updateTimeStamp = new SimpleStringProperty("Last update: ");
    }

    public void updatePieChart() {
        double[] vals = model.getDataValues();
        x.setValue(vals[0]);
        y.setValue(vals[1]);
        z.setValue(vals[2]);
        updateTimeStamp.setValue("Last updated: " + model.getLastUpdateTim
    }

    public DoubleProperty xProperty() {
        return x;
    }

    public DoubleProperty yProperty() {
        return y;
    }

    public DoubleProperty zProperty() {
        return z;
    }

    public StringProperty updateTimeStampProperty() {
        return updateTimeStamp;
    }
}
```




```java
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

    private PieChart.Data x = new PieChart.Data("X", 0);
    private PieChart.Data y = new PieChart.Data("Y", 0);
    private PieChart.Data z = new PieChart.Data("Z", 0);

    public PieChartView() {
    }

    public void init(PieChartViewModel pieChartViewModel) {
        this.viewModel = pieChartViewModel;

        x.pieValueProperty().bind(viewModel.xProperty());
        y.pieValueProperty().bind(viewModel.yProperty());
        z.pieValueProperty().bind(viewModel.zProperty());

        eventLabel.textProperty().bind(viewModel.updateTimeStampProperty());

        ObservableList<PieChart.Data> datas = FXCollections.observableArrayL
        pieChart.setData(datas);
    }

    public void onUpdateButton(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update

```java
public class PieChart

    private DoublePro
    private DoublePro
    private DoublePro
    private StringPro

    private Model mod

    public PieChartVi
        this.model =
        x = new Simpl
        y = new Simpl
        z = new Simpl
        updateTimeSta
    }

    public void updat
        double[] vals
        x.setValue(va
        y.setValue(va
        z.setValue(va
        updateTimeSta
    }

    public DoubleProp
        return x;
    }

    public DoubleProp
        return y;
    }

    public DoubleProp
        return z;
    }

    public StringProp
        return update
    }
}
```




```
public class PieChartView {

    @FXML
    Label eventLabel;

    @FXML
    PieChart pieChart;

    private PieChartViewModel viewModel;

                                    ieChart.Data("X", 0);
                                    ieChart.Data("Y", 0);
                                new PieChart.Data("Z", 0);



                                    iewModel pieChartViewModel) {
                                    artViewModel;

                                    ind(viewModel.xProperty());
                                    ind(viewModel.yProperty());
                                    ind(viewModel.zProperty());

                                    y().bind(viewModel.updateTimeStampProperty());

                                    t.Data> datas = FXCollections.observableArrayL
                                    s);


                                    n(ActionEvent actionEvent) {
        viewModel.updatePieChart();
    }
}
```

Update call done

```
public class PieChart

    private DoublePro
    private DoublePro
    private DoublePro
    private StringPro

    private Model mod

    public PieChartVi
        this.model =
        x = new Simpl
        y = new Simpl
        z = new Simpl
        updateTimeSta
    }

    public void updat
        double[] vals
        x.setValue(va
        y.setValue(va
        z.setValue(va
        updateTimeSta
    }

    public DoubleProp
        return x;
    }

    public DoubleProp
        return y;
    }

    public DoubleProp
        return z;
    }

    public StringProp
        return update
    }
}
```

Update

Benefits

- It provides separation of concerns. Tightly coupled, change resistant, brittle code causes all sorts of long-term maintenance issues that ultimately result in poor customer satisfaction with the delivered software. A clean separation between application logic and the UI will make an application easier to test, maintain, and evolve. It improves code re-use opportunities and enables the developer-designer workflow.
- It is a natural pattern for XAML platforms. The key enablers of the MVVM pattern are the rich data binding stack of the Silverlight platform, and dependency properties. The combination of these provides the means to connect a UI to a view model.
- It enables a developer-designer workflow. When the UI XAML is not tightly coupled to the code-behind, it is easy for designers to exercise the freedom they need to be creative and make a good product.
- It increases application testability. Moving the UI logic to a separate class that can be instantiated independently of a UI technology makes unit testing much easier.

- https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)

End of part 4