



ZK Training

MODULE 2-2

MVVM

Table of Contents

2

- Introduction
- Data Binding Basics
- Validation
- Converter
- Global Command Binding
- Passing Parameters
- Template binding
- ViewModel Communication

MVVM Introduction

3

Introduction

4

- **Concept**
 - View - markup declarations
 - Binder - the agent between View and ViewModel
 - BindComposer - initializes Binder and ViewModel
 - ViewModel - state object of View
- **A Quick Sample**

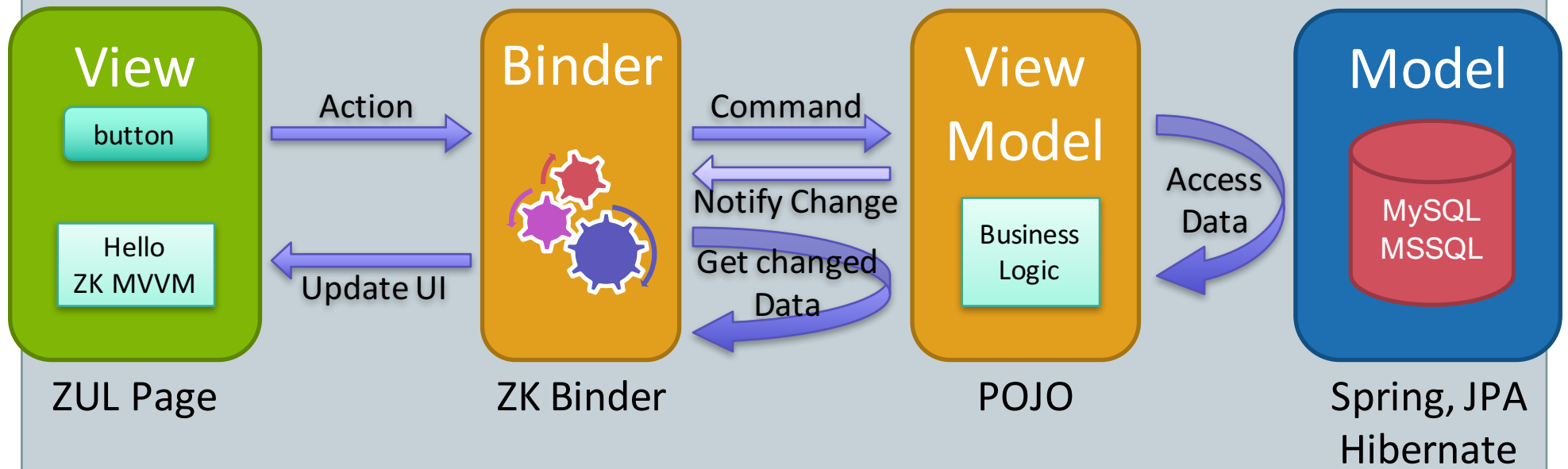
Introduction

5

- MVVM is an abbreviation of a design pattern named Model-View-ViewModel which originated from Microsoft WPF
- a variant of the famous MVC pattern
- This pattern has 3 roles: View, Model, and ViewModel.
 - The View and Model plays the same roles as they do in MVC

Detail Operation Flow

6



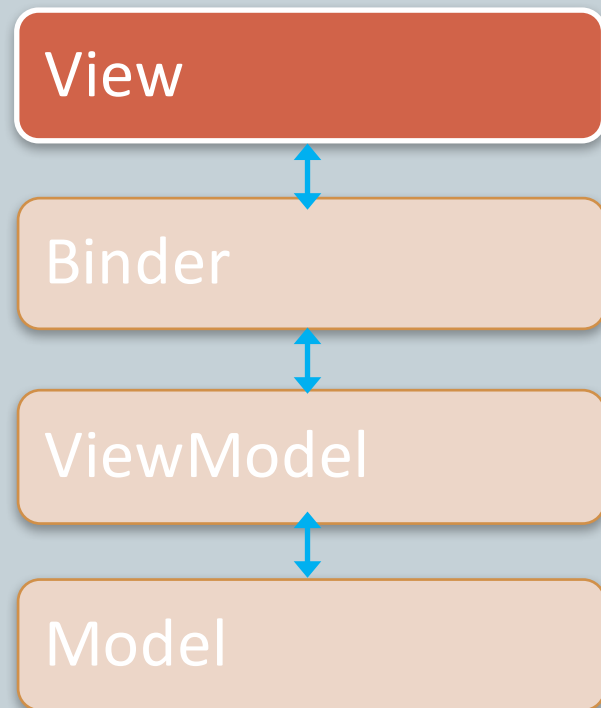
View

7

- What is View ?

- Apply the *org.zkoss.bind.BindComposer*
- Specify the ViewModel instance
- Use the ViewModel *ID* to reference ViewModel properties
- Use *annotations* to sync data and trigger methods in ViewModel

```
<window apply="org.zkoss.bind.BindComposer"
  viewModel="@id('vm') @init('pkg.LoginVM')">
  <label id="message" value="@load(vm.message)" />
  <label="Name" />
  <textbox value="@save(vm.name)" />
  <button label="Login" onClick="@command('login')" />
</window>
```

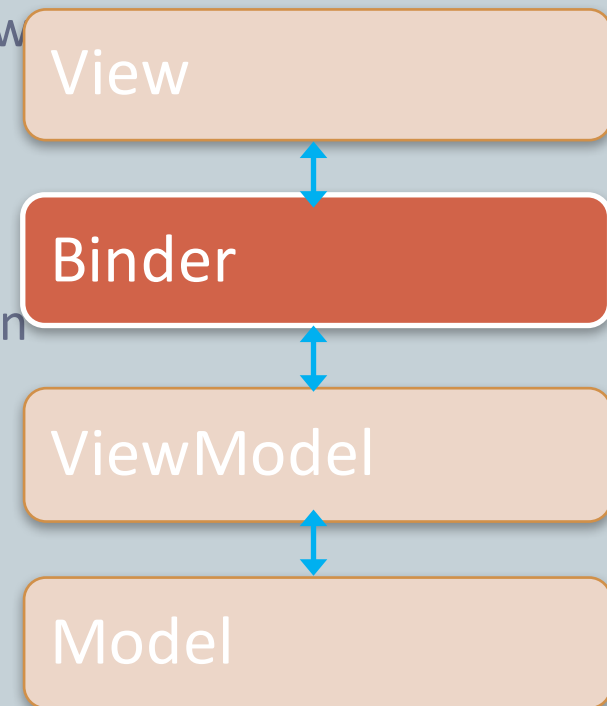


Binder

8

- **What is Binder?**

- Establish data-binding between the affected View components and ViewModel by parsing annotations in View
- Sync data between View and ViewModel
- Execute command (methods) in ViewModel upon events on the View

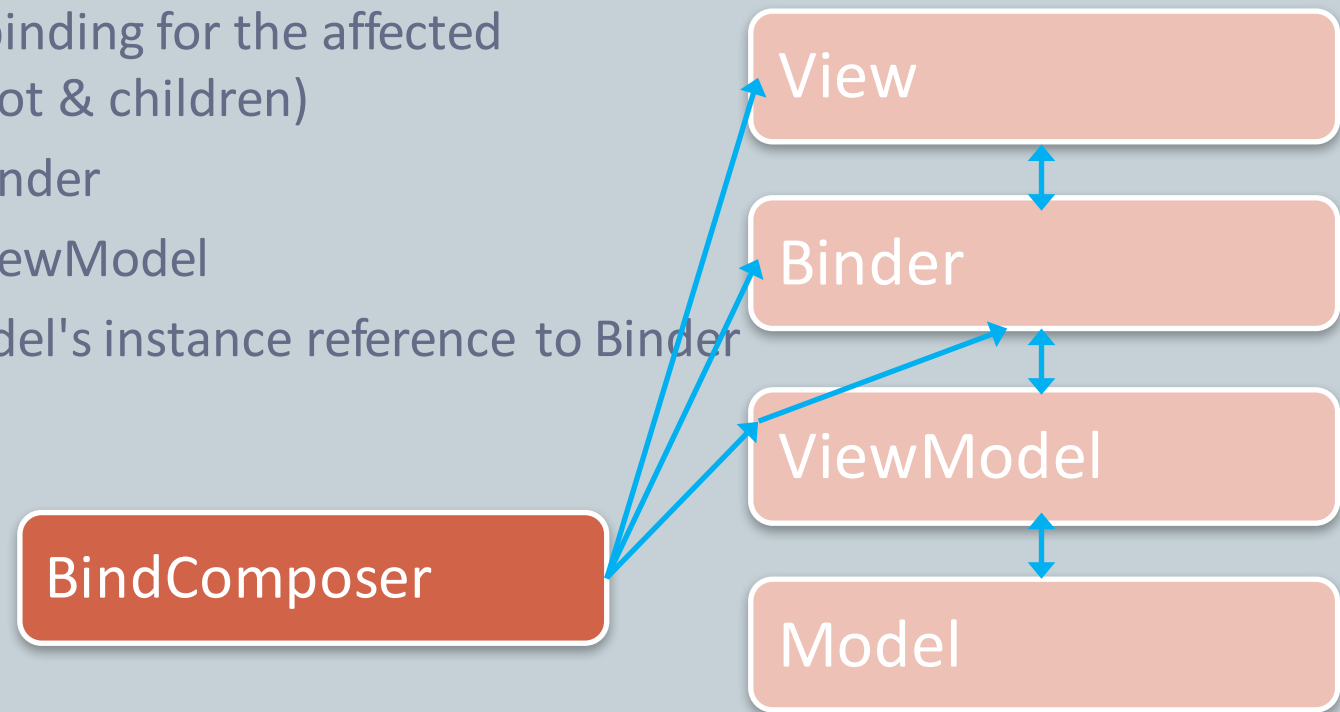


Binder Initialization

9

- What BindComposer does?

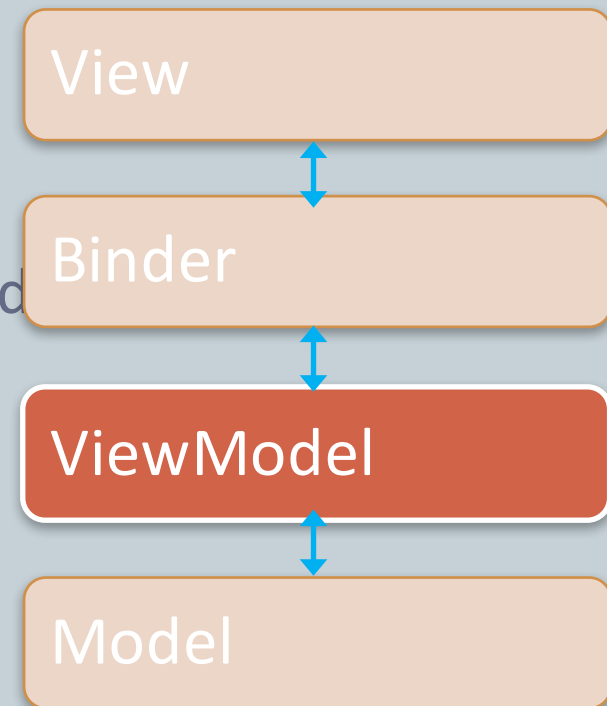
- Activates data-binding for the affected components (root & children)
- Instantiates a Binder
- Initializes the ViewModel
- Passes ViewModel's instance reference to Binder



View Model

10

- What is View Model?
 - A POJO
 - ✦ No need to extend a parent class
 - ✦ No need to Implement an interface
 - Has its properties accessible in annotated View components through Binder
 - Has its methods invoked by Binder
 - Notifies the Binder of changes in its properties

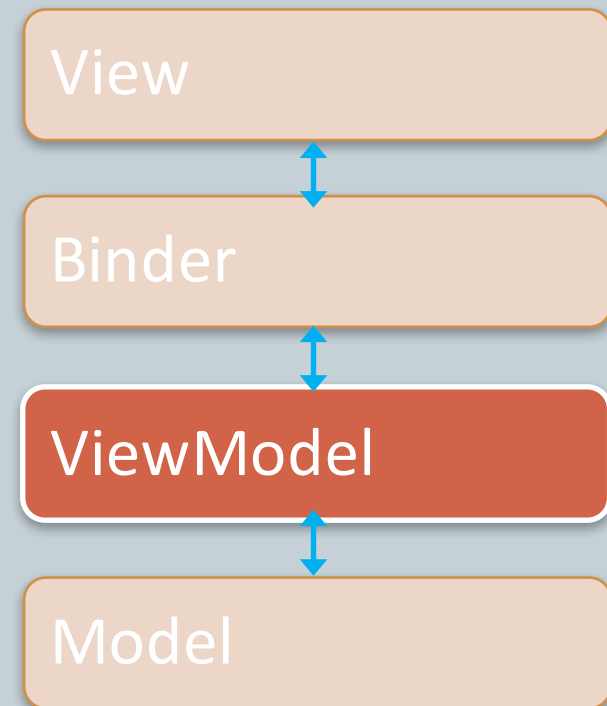


Specialties of View Model

11

- Has NO Reference to View
 - UI Event:
 - ✦ Binder invokes command methods (event handlers) in ViewModel
 - Data:
 - ✦ Binder keeps data in sync between View and ViewModel

ViewModel is Made Independent of View(s)



Strength of MVVM

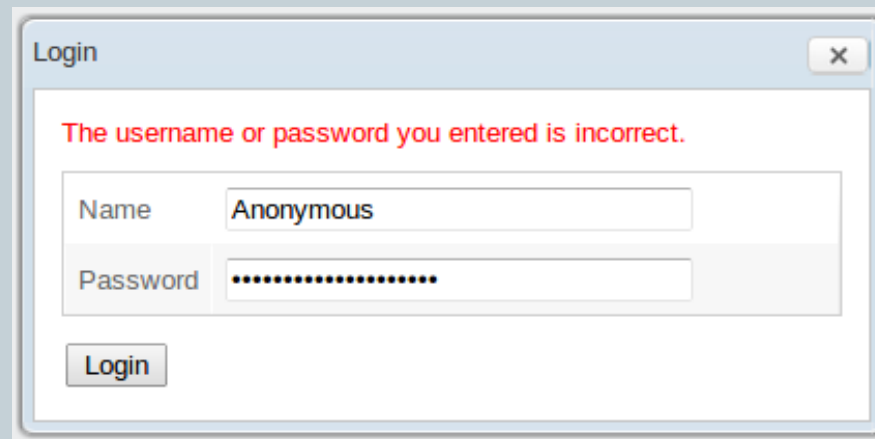
12

- **Loose coupling with View.**
 - UI design can be easily changed from time to time without modifying the ViewModel as long as the contract does not change.
- **Better reusability.**
 - It will be easier to design different views for different devices with a common ViewModel. For a desktop browser with a bigger screen, more information can be shown on one page; while for a smart phone with limited display space, designing a wizard-based step-by-step operation UI can be done without the need to change (much of) the ViewModel.
- **Better testability.**
 - Since ViewModel does not "see" the presentation layer, developers can unit-test the ViewModel class easily without UI elements.
- **It's suitable for design-by-contract programming.**
 - As long as the contract is made (what data to show and what actions to perform), the UI design and coding of ViewModel can proceed in parallel and independently. Either side will not block the other's way.

A Quick Example

13

- Say a login dialog - login.zul:



Login

The username or password you entered is incorrect.

Name Anonymous

Password

Login

that shows an error message if the name and password combination does not have a match in the database

Binding Relationship

14

```
public class LoginVM {  
    //javabean getter & setter  
    public void setMessage(String value) {...}  
    public String getMessage() {...}  
  
    public void setName(String value) {...}  
    public String getName() {...}  
  
    public void setPassword(String value) {...}  
    public String getPassword() {...}  
  
    @Command  
    @NotifyChange("message")  
    public void login() {  
        ...  
    }  
}
```

```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')">  
    <label id="message"  
        value="@load(vm.message)" />  
    <label="Name" />  
    <textbox value="@save(vm.name)" />  
    <label="Password" />  
    <textbox type="password"  
        value="@save(vm.password)" />  
    <button label="Login"  
        onClick="@command('login')" />  
</window>
```

Login

The username or password you entered is incorrect.

Name Anonymous

Password

Login

Data Binding

15

Property Binding

16

- 4 types
 - Init
 - ✦ load ViewModel's property to View once, not tracking the property
 - Load
 - ✦ load ViewModel's property to View (a component's attribute)
 - Save
 - ✦ save data from View (a component's attribute) to ViewModel's property
 - Bind
 - ✦ load + save
- Specify a binding expression at a component attribute
- `property.zul`

Property Binding – Load

17

- How component load VM properties?
 - *@load* is a ZUL element annotation
 - VM should provide a getter
 - A ViewModel's property is loaded to View when:
 - ✦ A component is rendered for the first time
 - ✦ the command method decorated with @NotifyChange finishes execution

```
public class LoginVM {  
    public String getMessage() {  
        return message;  
    }  
    @Command  
    @NotifyChange("message")  
    public void login() {  
        message = "... is incorrect.";  
    }  
}
```

```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')>  
    <label id="message"  
        value="@load(vm.message)" />  
</window>
```

Property Binding – Save

18

- How component save VM properties?
 - *@save* is a ZUL component annotation
 - VM should provide a setter
 - An input is saved to a ViewModel property when:
 - ✦ A specific event associated with the component is fired, e.g. onChange fired in textbox, intbox, spinner, etc.

```
public class LoginVM {  
    public void setName(String name) {  
        ...  
    }  
}
```

```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')>  
    <label value="name" />  
    <textbox value="@save(vm.name)" />  
</window>
```

Property Binding – Bind

19

- Effectively the same as @load + @save

```
public class LoginVM {  
    public void setName(String name) {  
        ...  
    }  
    public String getName() {  
        ...  
    }  
}
```

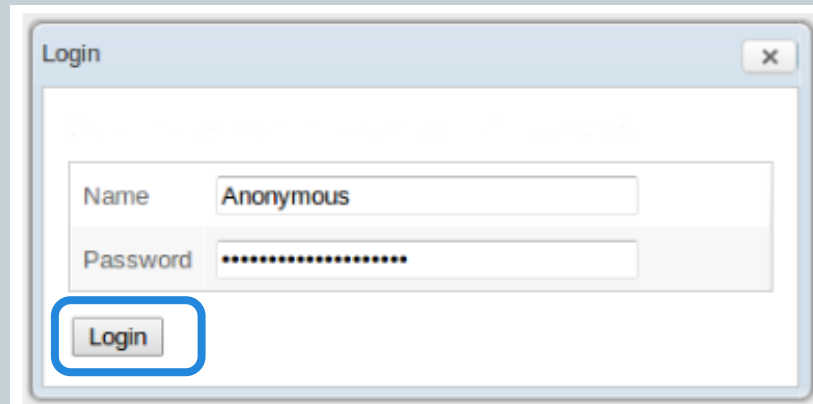
```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')>  
    <label value="name" />  
    <textbox value="@bind(vm.name)" />  
</window>
```

Property binding

20

- **Scenario**

- Suppose we want to enable the "Login" button only if user has entered their name and password:



- How do we accomplish this without having direct reference to the button component as we do in MVC?
- login.zul

Property binding

21

- dynamically set value to ANY component attribute via EL expression
- Sample Usage:
 - Set component value
 - Show/hide component
 - Enable/disable component
 - Set style
 - Load data collection

```
<button label="@load(vm.button_label)"
        disabled="@load(empty vm.show)"
        style="@load(vm.tempStyle)">
</button>
```

```
<window visible="@load(vm.visible)">
</window>
```

```
<listbox model="@load(vm.collection)"
          selectedItem="@bind(vm.selected)">
</listbox>
```

Property binding

22

- Making the Login button enable only if both name and password are entered

```
<window apply="org.zkoss.bind.BindComposer"
  viewModel="@id('vm') @init('pkg.LoginVM')>
  <label value="name" />
  <textbox value="@bind(vm.name)" />
  <label value="password" />
  <textbox type="password" value="@bind(vm.password)" />
  <button label="Login"
    disabled="@load(empty vm.name or empty vm.password)">
  </button>
</window>
```

Property binding - exercise

23

- Hello world – hello.zul
 - enter a name and show "hello [name]"

Command Binding

24

- Bind a component event to a method in ViewModel
 - `@command` in a `zul` \leftrightarrow `@Command` in a `ViewModel`
- Empty command
- Default command
- Pass parameters

Command Binding – notify change

25

- **Notify ViewModel state changes to View**
 - `@NotifyChange("oneProperty")`
 - `@NotifyChange({"property01", "property02"})`
 - `@NotifyChange("*")`
 - `@NotifyChange(".")` – notify a bean change
 - enabled by Default on setter methods

Command Binding – Notify Change

26

- Login – login.zul
 - After the method login() is executed, the binder will be informed that the value for message has been modified
 - The binder in turn will load the new value for vm.message in label

```
public class LoginVM {  
    @Command  
    @NotifyChange(*)  
    public void login() {  
        message = "...";  
    }  
}
```

```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')>  
    <label value="@load(vm.message)" />  
    <button label="Login"  
        onClick="@command('login')" />  
</window>
```

Command Binding – example

27

- Login – login.zul
 - Map user's action (ex: onClick) to a command
 - Annotate a method in ViewModel with the command name
 - Be aware of the capital differences between VM & ZUL "command" annotation!

```
public class LoginVM {  
    @Command("login")  
    public void dologin() {  
        ...  
    }  
}
```

```
<window apply="org.zkoss.bind.BindComposer"  
    viewModel="@id('vm') @init('pkg.LoginVM')>  
    <button label="Login"  
        onClick="@command('login')" />  
</window>
```

Exercise

28

- User login – login.zul
- Login button enabled when both fields are not empty
- Validation fails, show the error message
- Validation succeeds, redirect to index.zul
- Press enter to submit
- Press ESC to reset

ZK Shop - Login

Name:

Password:

➔ Login

Passing Parameters

29

- pass any object or value that can be referenced by EL on a ZUL to command
- `@command('commandName' ,
[arbitraryKey]=[EL-expression])`

Passing Parameters

30

- Receive parameters with `@BindingParam`
 - `commandParameter.zul`

```
<button label="Index" onClick="@command('showIndex',  
index=itemStatus.index)"/>
```

```
@Command  
public void showIndex(@BindingParam("index") Integer index) {  
    message = "item index: " + index;  
}
```

Collection Binding

31

- **Template**
 - Iteratively render data items
- **Selected Item**
 - Reference and operate on the data item selected
- **Change Template Dynamically**

Collection Binding – View & ViewModel

32

- Make the collection of objects available in the ViewModel instance
 - ✦ `ListModelList<String> items; //getter`
- Bind the collection to the model attribute
 - ✦ `model="@load(vm.items)"`

```
<listbox model="@load(vm.items)">
  <listhead>
    <listheader label="Item
List"/>
  </listhead>
</listbox>
```

```
public class CollectionBindingVM {
    private List<String> items;

    public List<String> getItems() {
        return items;
    }
}
```


Implicit Iteration Variable

33

- **each**
 - iteration object variable which references to each object of the model. We can use it to access an object's properties with dot notation, e.g. `each.name` .
- **forEachStatus**
 - iteration status variable. it's used to get iteration index by `forEachStatus.index` .
- Iteration variable can be overridden by “var”
 - `var=“book”`
 - `bookStatus`

Collection Binding – Template

34






- `<template name="name" var="variable" >`
 - ✦ declare a template to iterate through the collection
 - ✦ "var" holds the reference name to each individual object
 - ✦ when "var" is not specified, use the keyword "each"
 - ✦ a "name" must be given to reference the template

```
<listbox model="@load(vm.items)">
  <listhead>
    <listheader label="Item List"/>
  </listhead>
  <template name="model" var="item">
    <listitem>
      <listcell label="@load(item)" />
    </listitem>
  </template>
</listbox>
```

Exercise

35

- Display products with Grid – index.zul

	Name	Price	Quantity	Arrive Date	Buy
	Cookies	4.0	30	2015/09/05	1 <input type="button" value="Add"/>
	Toast	3.0	43	2015/09/03	1 <input type="button" value="Add"/>
	Chocolate	5.1	12	2015/08/29	1 <input type="button" value="Add"/>
	Butter	2.5	60	2015/09/02	1 <input type="button" value="Add"/>
	Milk	3.1	71	2015/09/04	1 <input type="button" value="Add"/>

Selection Handling

36

- collection.zul
- ViewModel
 - Declare a property selected of type String
- View
 - selectedItem holds reference to the item user selected
 - binds the attribute “selectedItem” to the property “selectedItem” in our ViewModel
 - Load the the selected item to a UI component to verify it

```
public class CollectionBindingVM {  
    String selectedItem;  
    //getter  
}
```

```
<listbox  
    selectedItem="@bind(vm.selectedItem)">  
...  
</listbox>  
Selected:  
<label value="@load(vm.selectedItem)"/>
```


Exercise

37

- Display shopping cart items with Listbox – index.zul
 - Add sample data
- Show product image for selected item

Name	Price	Amount	Subtotal	
Toast	3.0	1	\$ 3.00	<input type="button" value="x"/>
Cookies	4.0	1	\$ 4.00	<input type="button" value="x"/>

Total: \$ 7.00



Comment

Note for this order.

Children Binding

38

- Collection data to children
- Template for child rendering
- children.zul
- Can be replaced by <forEach>

Form Binding

41

- Form Binding Concept
- Implementation
 - A typical use case
- Binding Conditional Decoration
 - Save form data before, or after executing a command
- Dirty Status
 - Form data dirty status indicator

Form Binding

42

- **Usage scenario**
 - When using a persistence framework like JPA , Hibernate
 - A entity bean usually stores the user input data to be committed to the database
 - If you don't want the bean to contain invalid data, you need a temporary place like the bean to store user input for validation
 - Then store the valid data into the an entity bean to commit to a database
 - Cancel input in a half way, need to restore original data
 - Tracking dirty status
 - Move input data from a temporary place to the target bean

Form Binding Concept

43

- Property Binding

```
<textbox  
  value="@save(vm.user.name)"/>
```

onChange

```
public class SignupVM() {  
  User user;  
}
```

user input
saved

- Form Binding

```
<grid form="@id('fx')@load(vm.user)  
  @save(vm.user, before='submit')">  
  <textbox value="@save(fx.name)"/>
```

```
  ...  
</grid>
```

onChange

Proxy object

onClick

```
<button onClick="@command(submit)"/>
```

```
public class SignupVM{  
  User user;  
  @Command  
  public void submit()  
  {...}  
}
```

user input
saved before
submit is
executed

Form Binding

44

- **Benefit**
 - Keep the target bean clean
 - Easy to restore the original data
 - Track dirty status
 - store input data to the target bean at once, automatically

Form Binding

45

- ZK implicitly creates a proxy object for a bean
 - when a bean's getter is called, ZK also creates a proxy object for the property
- support these types - **Collections, Map, and POJO**
- For a class to be proxied
 - should provide default constructor
 - if implement hashCode() and equals(), need to use getter method

Form Binding – Implementation

46

- Create a User object in the ViewModel

```
public class SignupVM {  
  
    private User user;  
    private String confirmedPassword = "";  
  
    @Command  
    public void submit(){  
        Clients.showNotification("submit");  
    }  
}
```

Form Binding – Implementation

47

- 1. Give an id to the proxy object in form attribute with @id .
- 2. Specify ViewModel's property to be loaded with @load
- 3. Specify ViewModel's property to save and before which Command with @save
- 4. Bind component's attribute to proxy object's properties like you do in property binding.
- Signup.zul

```
<grid form="@id('fx') @load(vm.user) @save(vm.user, before='submit')">
  <rows>
    <row>
      Name:   <textbox value="@bind(fx.name)" />
    </row>
    <row>
      Password: <textbox value="@bind(fx.password) " type="password" />
    </row>
```

Conditional Binding

48

- Before

```
<doublebox value="@save(vm.income.wage, before='calcNetIncome')" />
```

- After

```
<doublebox value="@load(vm.income.net, after='calcNetIncome')" />
```

- Multiple Conditions

```
<doublebox value="@load(vm.income.net,  
  after={'getTaxRate','calcNetIncome'})" />
```

Form Binding – Dirty Status

49

[id]Status.dirty

- True if at least one form field is edited
 - ✦ when the proxy object has been modified
- false otherwise

Form binding - Exercise

50

- A sign up form
 - Show dirty status
 - Show target User value
 - signup.zul

Sign Up

*

Name:	<input type="text" value="John"/>
Password:	<input type="password"/>
Confirm Password:	<input type="password"/>

The value in User object

The value in User object	
Name:	
Password:	

Form binding – proxy object

51

- Sometimes we need to access the generated form proxy object
 - Not just invoking setter and getter by data binding
- Pass form proxy as a parameter to access in a command method
- `formProxy.zul`
 - add category
 - Remove category

Form binding – exercise

52

- **formProxy.zul**
 - add a category
 - Remove a category

Validation

53

- Validators:

- Property Validator
- Save Before Validator
- Dependent Property Handling
- Form Validator
- Dependent Property Validator in Form Binding

The screenshot shows a web form titled "New Delete Edit-in-Place". The form contains several fields with validation errors indicated by red text:

- Order ID**: A yellow warning icon is next to the field.
- Item**: A dropdown menu showing "GP-01".
- Quantity**: A text input with "0". A red error message "must be greater than 0" is displayed.
- Price**: A text input with "0.00". A red error message "must be greater than 0" is displayed.
- Total Price**: A text input with "0.00".
- Creation Date**: A date picker icon. A red error message "must not be empty" is displayed.
- Shipping Date**: A date picker icon. A red error message "must be at least 3 days later than creation date" is displayed.
- Note**: A large text area.

At the bottom of the form, there are "Save" and "Cancel" buttons.

Property Validator

54

- Apply with property save binding
- Invoked before saving a value to a ViewModel
- use it by a getter method or FQCN
- validator.zul

Property Validator

55

- Create & use a Validator
 - Extends `org.zkoss.bind.validator.AbstractValidator`
 - ✦ provides `addInvalidMessage()`
 - Override `validate()`
 - ✦ Get user input value from `ValidationContext`
 - Add custom invalid message to validation context
 - *`validator.zul / org.zkoss.training.mvvm.EmailValidator`*
 - Apply it with a `@save / @bind`

Property Validator

56

- Show a validation message
 - `validationMessage` object
 - ✦ Initialize it first before using
 - ✦ A map that holds validation message as key-value pairs
 - ✦ Default message key: validated target component ID
 - ✦ Can specify a custom key
 - Load a validation message with a key
 - ✦ `@load(validationMessageId[comp_id])`
 - ✦ `@load(validationMessageId[myKey])`
 - `validator.zul`

Property Validator

57



- Used with conditional binding
- validate before the command *saveOrder* is executed
- *saveOrder()* will not be executed if validation fails
- orderManagement.zul

```
<doublebox id="dbx"  
    value="@bind(vm.selected.price, before='saveOrder')  
        @validator(vm.priceValidator)" />  
<button label="save" onClick="@command('saveOrder')" />
```

Property Validator

58

- Validate multiple fields
- need another property's value to validate the current property
- Both properties must be subjected to the same conditional binding (e.g., before='saveOrder') to share the same ValidationContext
- Shipping date must later than creation date
 - `orderManagement.zul / ShippingDateValidator`

Creation Date	<input type="text"/>	 must not be empty
Shipping Date	<input type="text"/>	 must be at least 3 days later than creation date

Exercise

59

- `signup.zul`
 - `nameConflictValidator` – to avoid signup a duplicate account

Sign Up

Name:

zk

Password:

Confirm Password:

User name already exists

Submit

Property Validator - form

60

- Assign a validator on a form binding
- Executed before input value is saved to the form proxy object
 - `orderForm.zul / @validator(vm.quantityValidator)`

Form Validator – validate multiple fields

61

- Assign validator to a form instead of a component
 - The dependent must be validated in form since that's where the save-before condition is applied

```
<groupbox id="gpbox"  
  form="@id('fx') @load(vm.selected) @save(vm.selected, before='saveOrder')  
  @validator(vm.shippingDateValidator)">  
</groupbox>
```

Converters

62

- Built-in converter
- Custom converter
- Implicit converter

Converters

63

- performs two way conversion between ViewModel's property and UI component attribute
- converts data to component attribute when loading ViewModel's property to components
- converts back when saving to ViewModel.
- It's quite common requirement to display data in different format in different context.
 - developers can achieve this without actually changing the real data in a ViewModel.

Built-in Converter – formattedNumber

64

- *org.zkoss.bind.converter.FormatedNumberConverter*
- `ctx.getConverterArg("format")`
 - retrieve the format as specified in markup
 - use the retrieved format against the input float to return a string
- Sample Usage – converter.zul

```
<listcell label="@load(order.price)
  @converter('formattedNumber',format='###,##0.00')"/>
```

Built-in Converter – formattedDate

65

- *org.zkoss.bind.converter.FormatedDateConverter*
- `ctx.getConverterArg("format")`
 - use the retrieved format against the input string to return a date
- Sample Usage – `orderManagement.zul`

```
<listcell label="@load(order.creationDate)  
  @converter('formattedDate', format='yyyy/MM/dd')"/>
```

Custom Converter

66

- An implementation of *org.zkoss.bind.Converter*
- **coerceToUi**
 - invoked before loading object property to UI component
 - converting an object property to be displayed on UI
- **coerceToBean**
 - invoked before saving UI input to object property
 - converting input from UI to an object property

Implicit converter

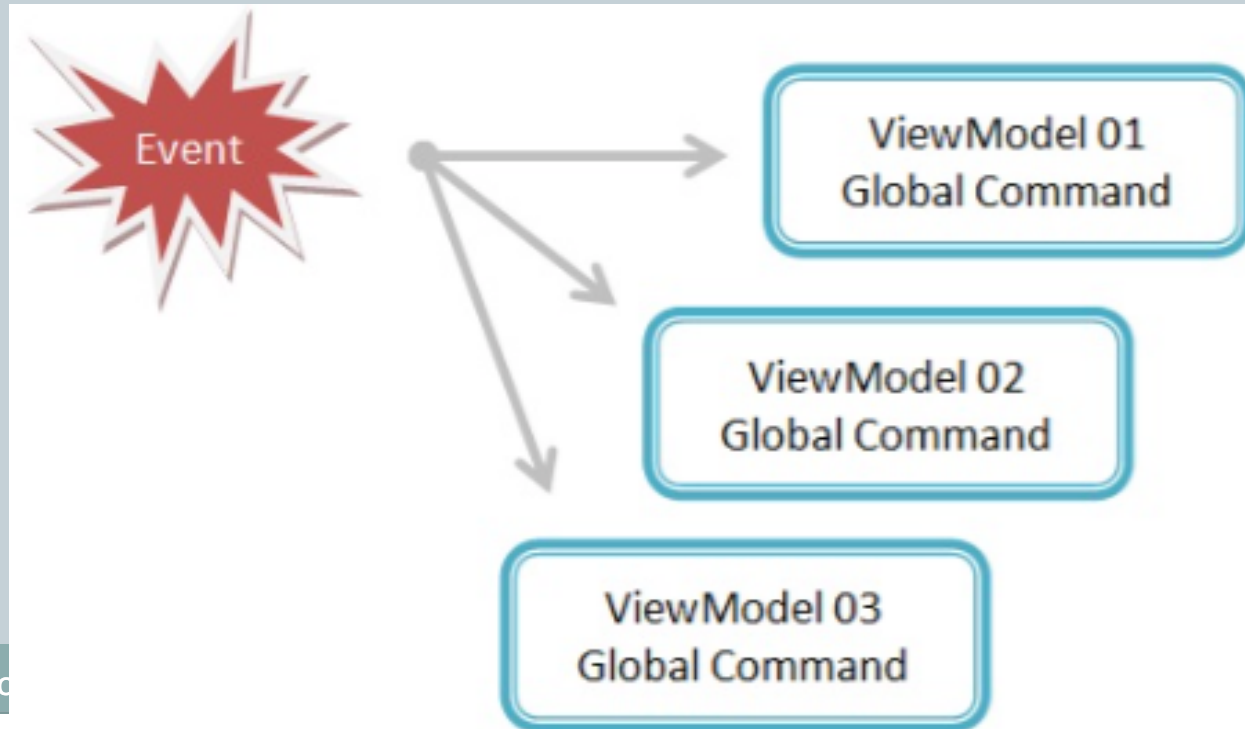
67

- ZK works behind the scenes for our convenience.
 - `org.zkoss.bind.converter.sys.AbstractListModelConverter`
 - ✦ Converts raw data collection type to ListModel
 - data collection wrapped in ListModel is updated automatically according to data value changes in UI
 - `org.zkoss.bind.converter.sys.ListBoxSelectedItemConverter`
 - ✦ Converts selected ListItem to bean
 - `org.zkoss.bind.converter.sys.ListBoxSelectedIndexConverter`
 - ✦ Converts selected ListItem index to bean

Global Command Binding

68

- all matched global command in all ViewModels in the scope will be executed
- Default scope is “desktop”



Global Command Binding

69

- Usage
 - Notify another ViewModel in the same scope
 - One to many communication
 - `globalCommandParameter.zul`
- Can use with `@command`
 - Validation fails will stop local command, then a global command, too

Global Command Binding - exercise

70

- globalCommandParameter.zul
 - Add an item

Global Command Binding - programmatically

71

- Used when
 - Trigger a command depending on runtime value
 - Trigger a command inside another command, rather than in a `zul`

Global Command Binding - exercise

72

- `globalCommandParameter.zul`
 - Clear input item after adding

Global Command Binding

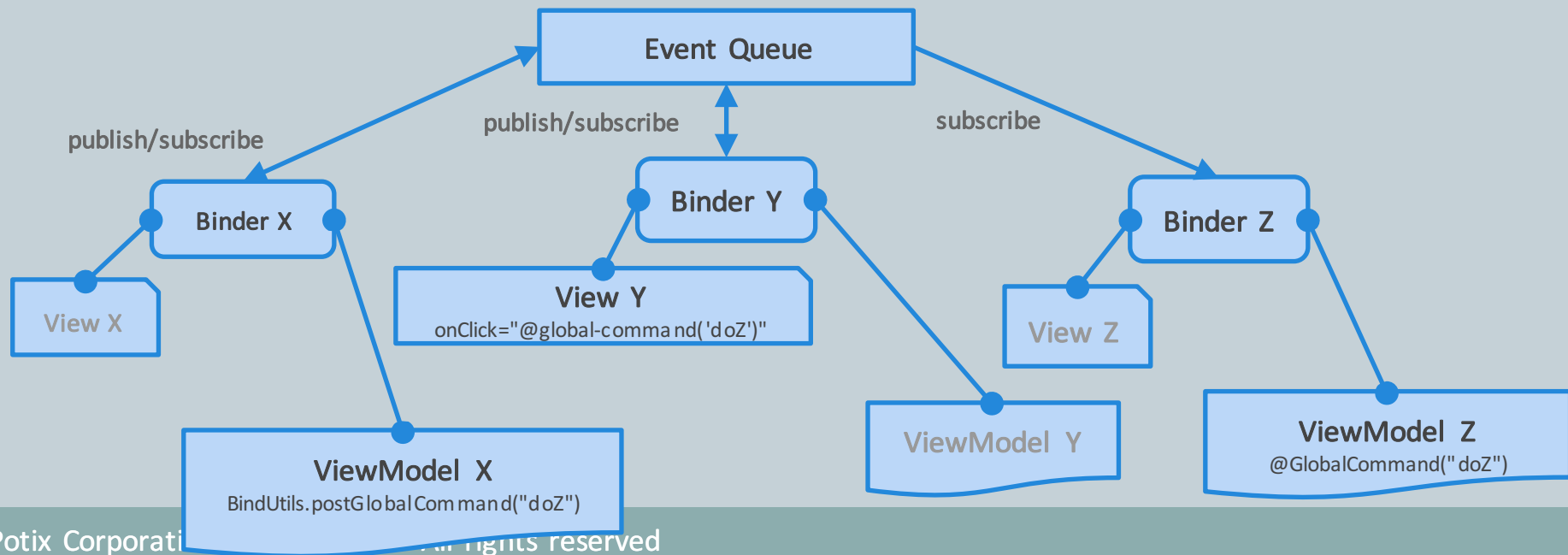
73

- One-to-many communication
 - globalCommandParameter.zul
 - Show and hide

Global Command – Under the hood

74

- All binders subscribe to a desktop scope Event Queue
- An event bound to a GlobalCommand is published to the Event Queue
 - (ViewModel X, View Y)
- All binders receive such events and check all GlobalCommands in VM for a match
 - (Binder X, Y, Z)
- If a match is found, the GlobalCommand is executed
 - (Binder Z finds doZ in ViewModel Z and doZ gets executed)



Passing Parameters – context objects

75

- Retrieve context objects by @ContextParam

```
public void init(@ContextParam(ContextType.SESSION) Session session) {...}
```

- Can be used on those methods applied with @Init, @Command, @GlobalCommand
- Other parameter annotations
 - @QueryParam
 - @HeaderParam
 - @CookieParam
 - @ExecutionParam
 - @ExecutionArgParam
 - @ScopeParam
 - @SelectorParam
 - @Default


Exercise

76

- *@global-command('addToCart')*

Shopping Cart

Log Off

	Name	Price	Quantity	Arrive Date	Buy		Name	Price	Amount	Subtotal		
	Cookies	4.0	30	2015/09/06	1	<div><div></div><div></div><div>Add</div></div>	Cookies	4.0	1	\$ 4.00	<div><div></div></div>	
							<div>Submit</div>	<div>Clear</div>	Total:	\$ 4.00		
							Comment					
							Note for this order.					

Template Binding

77

- Swap template while in-place editing
- Replace by <apply>

Order Management

Delete Edit-in-Place

ID	Item	Quantity	Price	Creation Date	Shipping Date
00001	RX-79	6	85.57	2012/06/20	2012/06/23
00002	RX-79B	4	34.45	2012/06/20	2012/06/23
00003	MSZ-007	3	41.15	2012/06/20	2012/06/23
00004	GP-01	1	149.29	2012/06/20	2012/06/23

Order Management

Delete Edit-in-Place

ID	Item	Quantity	Price	Creation Date	Shipping Date
00001	RX-79	6	85	Jun 20, 2012	Jun 23, 2012
00002	RX-79B	4	34	Jun 20, 2012	Jun 23, 2012
00003	MSZ-007	3	41	Jun 20, 2012	Jun 23, 2012
00004	GP-01	1	149	Jun 20, 2012	Jun 23, 2012
00005	GP-03	9	135	Jun 20, 2012	Jun 23, 2012

Composer to ViewModel Communication

82

- Posting a Command from a Composer to a ViewModel
- In a composer
 - `BindUtils.postGlobalCommand(null, null, "addCart", null);`
 - ✦ Queue name, scope, command name, parameters(Map)
- A ViewModel receives it with a global command
- `communication.zul`

ViewModel to Composer Communication

83

- Posting a Command from a ViewModel to a Composer
- In a ViewModel
 - Nothing to do
 - Add a global command binding in a zul
- In a composer
 - Subscribe the same event queue as the binder to receive GlobalCommand
 - Cast event type if is GlobalCommandEvent, check if the `getCommand()` result equals `CLOSE_DIALOG`; if true, close the Edit Profile dialog
- `communication.zul`

Documents

84

- [Data Binding Cheat Sheet](#)
- [MVVM Reference](#)

Q & A