Android Forms Documentation

Release 1.0.8

Eddilbert Macharia

Feb 07, 2018

Contents:

This library makes it easy to deal with views, especially if dealing with a large amount of views.

It provide a consistent and fluent way of setting, validating, retrieving and saving values from views.

With this library you not have to change or alter you layout in anyway, but you will be able to work with multiple forms on multiple fragments via *FormCollection*.

The library also comes an inbuilt *Data populator* to use when developing. This comes in handy when you want to prepopulate you form with sample data.

For validation the Validation is used.

CHAPTER 1

install

using Maven.

```
<dependency>
  <groupId>com.eddmash</groupId>
  <artifactId>android-form</artifactId>
  <version>1.0.7</version>
  <type>pom</type>
</dependency>
```

Using gradle

```
compile 'com.eddmash:android-form:1.0.7'
```

CHAPTER 2

Usage

To use this library is very easy

- Create *Form* class

```
 private class BasicForm extends Form {
        public BasicForm() {
            super();
        }

        public BasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and␣
↪the values
        }
}
```

- Create form object

```
BasicForm form = new BasicForm();
```

- Add *Field* to *FormInterface*

```
// get the views from the layout
Spinner genderSpinner = (Spinner) view.findViewById(R.id.gender);
EditText fName = (EditText) view.findViewById(R.id.firstname);
EditText phone_number = (EditText) view.findViewById(R.id.phone_number);

// add views to the form
form.addField("gender", genderSpinner);
form.addField("firstname", fName);
```

```
form.addField("phone_number", fName);

// add validation check
form.addCheck(new NotEmptyCheck(gender,"Gender cannot be blank"));
form.addCheck(new NotEmptyCheck(fName,"Firstname cannot be blank"));
```

- Run *validation* and get the *values*.

```
if(form.isValid()){
    form.getValues()// returns a map of where key is the name of the field and the␣
↪values

}else{

     LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
     errorSpace.removeAllViews();// clear space first

     ErrorRenderer errorRenderer = new ErrorRenderer(this, form.getValidator());
     errorRenderer.render(errorSpace);
}
```

- *Save* form *values*

```
if(form.isValid()){
    try{
        form.save() // save
    } catch (FormException e) {
        e.printStackTrace();
    }

}else{

    LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
    errorSpace.removeAllViews();// clear space first

    ErrorRenderer errorRenderer = new ErrorRenderer(this, form.getValidator());
    errorRenderer.render(errorSpace);
}
```

2.1 Form Collection

This library also supports dealing with multiple forms at once.

A *FormCollection* class accepts *InnerForm*

This makes it possible:

- **for forms to depend on each other**
  - that is a form A cannot be validated before form B is validated.
  - that is a form A cannot be saved before form B is saved.
- multiple forms get validated together.
- multiple forms get saved together.

2.1.1 Usage

To use this library is very easy

- Create *InnerForm* class

In this example, the DependOnBasicForm requires the BasicForm to have been validated on the validation stage and be saved on saving stage.

```
 private class BasicForm extends InnerForm {
        public BasicForm() {
            super();
        }

        public BasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public String getIdentifier() {
            return "basic_form_id";
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and␣
↪the values
        }
}

 private class DependOnBasicForm extends InnerForm {
        public DependOnBasicForm() {
            super();
        }

        public DependOnBasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and␣
↪the values
        }


        @Override
        public String getIdentifier() {
            return "depend_on_basic_form_id";
        }

        @Override
        public String[] requires() {
            return new String[]{"basic_form_id"}; // make this form depend on the␣
↪basicform
        }
}
```

- Create *FormCollectionInterface* object

```
FormCollection formCollection = new FormCollection();
```

- Add *InnerForm* to *FormCollection*.

```
// create instance of inner form
BasicForm basicform = new BasicForm();
DependOnBasicForm dependOnBasicForm = new DependOnBasicForm();

// add inner form to collection
formCollection.addForm(basicform);
formCollection.addForm(dependOnBasicForm);
```

- Run *validation* and get the *values*.

```
if(formCollection.isValid()){
    formCollection.getValues()// returns a map of where key is the name of the field␣
↪and the values

}else{

     LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
     errorSpace.removeAllViews();// clear space first

     ErrorRenderer errorRenderer = new ErrorRenderer(this, formCollection.
↪getValidator());
     errorRenderer.render(errorSpace);
}
```

- *Save* form *values*

```
if(formCollection.isValid()){
    try{
        formCollection.save() // save
    } catch (FormException e) {
        e.printStackTrace();
    }

}else{

    LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
    errorSpace.removeAllViews();// clear space first

    ErrorRenderer errorRenderer = new ErrorRenderer(this, formCollection.
↪getValidator());
    errorRenderer.render(errorSpace);
}
```

2.2 Data Population

When developing, especially when working with form we need to prepopulate the form with sample data to ease up our development.

The *faker* is meant to take care of this for us.

Lets populate our *basic form* form from earlier with sample data.

```
form // the basicform from earlier example

// create a populator object
DummyDataPopulator populator = new DummyDataPopulator();

// format how the data should look
// this will generate data with the format a mobile number of we use here in kenya
// +254 722 472 447
populator.setFieldPopulator("phone_number", new Telephone().setFormat("(+###) ### ###
↪###"));


// tell populator to populate the form.
// it will use the provider we set above for the phone number instead of the default␣
↪one which
// would be a random set of numbers
populator.populate(form);

// you could also populate single field on its own
populator.populate(form.getField("gender"));
```

The kind of data that is generated depends on the type of view and the name of the field. e.g. if a view is edittext of with inputype of number then the populator will generated numbers for that particular view.

2.3 Javadoc

2.3.1 com.eddmash.form

Form

public abstract class **Form** implements *FormInterface*

Constructors

Form

public **Form**()

Form

public **Form**(ValidatorInterface *validator*)

Methods

addCheck

public void **addCheck**(CheckInterface *check*)

addField

public void **addField**(String *colName*, View *view*)

addField

public void **addField**(*FieldInterface field*)

disableCheck

public void **disableCheck**(CheckInterface *check*)

getErrors

public Map<String, List> **getErrors**()

getField

public *FieldInterface* **getField**(String *fieldName*)

getFields

public Map<String, *FieldInterface*> **getFields**()

getIdentifier

public String **getIdentifier**()

getValidator

public ValidatorInterface **getValidator**()

getValue

public Object **getValue**(String *fieldName*)

getValues

public Map<String, Object> **getValues**()

isValid

public boolean **isValid**()

removeField

public void **removeField**(String *colName*)

removeField

public void **removeField**(*FieldInterface field*)

setData

public void **setData**(Map *data*)

setValue

public void **setValue**(String *fieldName*, Object *value*)

validate

public void **validate**()

FormAwareInterface

public interface **FormAwareInterface**

Any class the implements this interface gets the form passed to it via the *setForm(FormInterface)*.

This class come in handy when creating validation checks that need to be away of the form, especially when performing form wide validations like on the *FormInterface.validate()* method.

A good examples is the *Field* which implements this interface.

The following examples how this interface can be used to create form wide check.

```
public class BasicForm extends Form {

     @Override
     public void validate() {
         try {
             addCheck(new PasswordCheck(password1Edittext, password2Edittext,
             "Password should match"));
         } catch (FormException e) {
             e.printStackTrace();
         }
     }

     @Override
     public void save() throws FormException {

     }

}
```

A sample check that uses the FormAwareInterface interface

```
class PasswordCheck extends CheckSingle implements FormAwareInterface {
     private final EditText view;
     private final String errorMsg;
     private FormInterface form;

     public PasswordCheck(EditText view, String errorMsg) {
         this.view = view;
         this.errorMsg = errorMsg;
     }

     @Override
     public boolean run() {
         try {
             Map val = form.getValues();
             Object pass1 = val.get("password_1");
             Object pass2 = val.get("password_2");
             if (pass1.equals(pass2)) {
                 return true;
             }
         } catch (FormException e) {
             e.printStackTrace();
         }
         return false;
     }

     @Override
     public String getErrorMsg() {
         return errorMsg;
     }
     @Override
     protected TextView getView() {
         return view;
     }
     @Override
     public void setForm(FormInterface form) {
         this.form = form;
     }
 }
```

Methods

setForm

void **setForm**(*FormInterface form*)

The form that will hold this class is passed in.

Parameters

- **form** –

FormException

public class **FormException** extends Exception

Constructors

FormException

public **FormException**(String *message*)

FormInterface

public interface **FormInterface**

Methods

addCheck

void **addCheck**(CheckInterface *check*)
Add a validation check.

Parameters

- **check** –

addField

void **addField**(*FieldInterface field*)

addField

void **addField**(String *colName*, View *view*)

disableCheck

void **disableCheck**(CheckInterface *check*)
Disable a validation check

Parameters

- **check** –

getErrors

Map<String, List> **getErrors**()
Returns all the errors on the form after validation.

Returns the key is the form identifier and the values is a list of all the validation errors related with the form.

getField

FieldInterface **getField**(String *fieldName*)

getFields

Map<String, *FieldInterface*> **getFields**()

getIdentifier

String **getIdentifier**()
A unique identifier for this form.

getValidator

ValidatorInterface **getValidator**()
The validator this form will be using to validate the inner forms.

getValue

Object **getValue**(String *fieldName*)
Returns the value a particular field.

The returned values depends on the *field* some fields the values is a string whilst others its a list of string.

Consult the specific *field* to get the returned value.

Parameters

- **fieldName** –

Throws

- ***FormException*** –

getValues

Map<String, Object> **getValues**()
Return the values of each field on the form.

The returned values depends on the *field* some fields the values is a string whilst others its a list of string.

Consult the specific *field* to get the returned value.

Throws

- ***FormException*** –

Returns a map, where keys a field identifier used when adding field to form and values are the fields respective values.

isValid

boolean **isValid**()
This is the entry point for form validations.

It firsts invokes the *validate()* to get the form wide validation .

It the tells the validator to run the validation check.

Returns true only if the validation checks passed.

removeField

void **removeField**(String *replace*)

removeField

void **removeField**(*FieldInterface field*)

save

void **save**()
This is where you should put you saving logic.

throw FormException if validation fails.

Throws

- ***FormException*** –

setData

void **setData**(Map *data*)

setValue

void **setValue**(String *fieldName*, Object *value*)
Set value for a specific.

Parameters

- **fieldName** – the identifier to use to locate the field being set.
- **value** – the value being set, this depends on specific *field* .consult specific *field* to find expected value.

validate

void **validate**()
This is the right place to perform form wide validations. That is validating fields against each other, also validate against parent form fields.

At this point you have access to the getValues() of both parent form and current form you can use this values to compare against.

The recommend approach is to a create check that implements FormAwareInterface and add it to the validator.

This method is invoked before the field specific validations have been run.

2.3.2 com.eddmash.form.collection

FormCollection

public class **FormCollection** implements *FormCollectionInterface*
Use this class when you want to deal on multiple forms all at once.

Constructors

FormCollection

public **FormCollection**()

FormCollection

public **FormCollection**(ValidatorInterface *validator*)

Methods

addForm

public void **addForm**(*InnerFormInterface* *form*)

getForm

public *InnerFormInterface* **getForm**(String *identifier*)

getValidator

public ValidatorInterface **getValidator**()

isValid

public boolean **isValid**()

removeForm

public void **removeForm**(*InnerFormInterface* *form*)

save

public boolean **save**()

FormCollectionInterface

public interface **FormCollectionInterface**
Use this class when you want to deal on multiple forms all at once.

Methods

addForm

void **addForm**(*InnerFormInterface* *form*)
Add a form into the collection.

Parameters

- **form** –

Throws

- ***FormException*** –

getForm

InnerFormInterface **getForm**(String *identifier*)
Get a form in the collection by its identifier.

Parameters

- **identifier** –

Throws

- ***FormException*** –

getValidator

ValidatorInterface **getValidator**()
The validator this collection will be using to validate the inner forms.

This is basically a parent validator which calls the individual validators bound to each of the inner forms.

isValid

boolean **isValid**()
Entry point for validation of all the innerforms on this collection.

Runs validation for each of the inner forms.

removeForm

void **removeForm**(*InnerFormInterface* *form*)
REmove a from the collection.

Parameters

- **form** –

save

boolean **save**()
The entry point of form saving.

This method calls the save() of each innerform attached to this collection

Wrap this method in a transaction to ensure if any of the inner form fails to save, All the other innerforms arent saved. for consistency sake.

Throws

- ***FormException*** –

InnerForm

public abstract class **InnerForm** extends *Form* implements *InnerFormInterface*
This is basically *form*that has the capability of being used with a form collection.

Fields

form

protected *FormCollectionInterface* **form**

Constructors

InnerForm

public **InnerForm**()

InnerForm

public **InnerForm**(ValidatorInterface *validator*)

Methods

getParent

public *FormCollectionInterface* **getParent**()

setParent

public void **setParent**(*FormCollectionInterface form*)

InnerFormInterface

public interface **InnerFormInterface** extends *FormInterface*

This is basically *form*that has the capability of being used with a form collection.

This form also has the added capability of depending on another form thats in the same collection as the one it belongs to.

Methods

getParent

FormCollectionInterface **getParent**()

The collection form in which this form belongs to.

requires

String[] **requires**()

An string array of other inner forms that this form should depend on i.e. those forms should be validated before this during the validation stage and should be saved before this is saved.

2.3.3 com.eddmash.form.faker

Callback

public abstract class **Callback**

Methods

invoke

public abstract String **invoke**()

DummyDataPopulator

public class **DummyDataPopulator** implements *PopulatorInterface*

This is a minimalistic go at data faker.

This intention is to populate the FormInterface and FieldInterfaces.

Constructors

DummyDataPopulator

public **DummyDataPopulator**()

Methods

populate

public void **populate** (*FormInterface* *form*)

populate

public void **populate** (*FieldInterface* *field*)

setFieldProvider

public void **setFieldProvider** (String *name*, *ProviderInterface* *provider*)

FakerException

public class **FakerException** extends Exception

Constructors

FakerException

public **FakerException** ()

FakerException

public **FakerException** (String *message*)

Guess

class **Guess**

Constructors

Guess

Guess (*PopulatorInterface* *populator*)

Methods

guess

public String **guess** (String *name*, View *view*)

PopulatorInterface

public interface **PopulatorInterface**
Its responsible for populating the *FormInterface* or *FieldInterface* provided.

The populator uses *providers* to populate each field presented to the populator.

Methods

populate

void **populate**(*FormInterface form*)
Tell the populator to start the population on the specified form.

Parameters

- **form** –

Throws

- ***FormException*** –

populate

void **populate**(*FieldInterface field*)
Tell the populator to start the population on the specified field.

Parameters

- **field** –

Throws

- ***FormException*** –

setFieldProvider

void **setFieldProvider**(String *name*, *ProviderInterface provider*)
Set the provider to use the populator a specific field.

Parameters

- **name** – the name of the field that will use the provider given.
- **provider** – the provider to use instead of the default onces.

2.3.4 com.eddmash.form.faker.provider

Company

public class **Company** extends *Provider*
Created by eddmash on 12/20/17.

Fields

NAME

String **NAME**

SUFFIX

String **SUFFIX**

type

String **type**

Methods

generate

public String **generate**()

getCompany

public *Company* **getCompany**()

getCompanySuffix

public *Company* **getCompanySuffix**()

Coordinates

public class **Coordinates** extends *Provider*

Fields

LATITUDE

String **LATITUDE**

LONGITUDE

String **LONGITUDE**

Constructors

Coordinates

public **Coordinates**()

Methods

generate

public String **generate**()

getLatitude

public *ProviderInterface* **getLatitude**()

getLongitude

public *ProviderInterface* **getLongitude**()

DateProvider

public class **DateProvider** extends *Provider*

Fields

TIME_NOW

public static final String **TIME_NOW**

TODAY

public static final String **TODAY**

dateFormat

protected String **dateFormat**

timeFormat

protected String **timeFormat**

Constructors

DateProvider

public **DateProvider**()

Methods

generate

public String **generate**()

getDate

public String **getDate**(String *format*)

getTime

public *ProviderInterface* **getTime**()

setDateFormat

public *DateProvider* **setDateFormat**(String *dateFormat*)

setTimeFormat

public *DateProvider* **setTimeFormat**(String *timeFormat*)

timeNow

public String **timeNow**()

today

public String **today**()

InternetProvider

public class **InternetProvider** extends *Provider*

Fields

DOMAIN

public static final String **DOMAIN**

EMAIL

public static final String **EMAIL**

TLD

public static final String **TLD**

Constructors

InternetProvider

public **InternetProvider**()

Methods

generate

public String **generate**()

setType

public *InternetProvider* **setType**(String *type*)

LocationsProvider

public class **LocationsProvider** extends *Provider*

Fields

ADDRESS

public static final String **ADDRESS**

CITY

public static final String **CITY**

COUNTRY

public static final String **COUNTRY**

Constructors

LocationsProvider

public **LocationsProvider**()

Methods

generate

public String **generate**()

getCity

public *LocationsProvider* **getCity**()

LoremProvider

public class **LoremProvider** extends *Provider*

Methods

generate

public String **generate**()

getWord

public *ProviderInterface* **getWord**()

getWords

public *ProviderInterface* **getWords**()

getWords

public *ProviderInterface* **getWords**(int *noOfWords*)

Person

public class **Person** extends *Provider*

Fields

EMAIL

public static final String **EMAIL**

FIRSTNAME

public static final String **FIRSTNAME**

LASTNAME

public static final String **LASTNAME**

NAME

public static final String **NAME**

PROVIDER_NAME

public static final String **PROVIDER_NAME**

Constructors

Person

public **Person**()

Methods

generate

public String **generate**()

getFirstName

public *ProviderInterface* **getFirstName**()

getFirstName

public *ProviderInterface* **getFirstName** (String *gender*)

getFullName

public *ProviderInterface* **getFullName** ()

getFullName

public *ProviderInterface* **getFullName** (String *gender*)

getLastName

public *ProviderInterface* **getLastName** ()

getLastName

public *ProviderInterface* **getLastName** (String *gender*)

setGender

public *Person* **setGender** (String *gender*)

setType

public *Person* **setType** (String *type*)

Provider

public abstract class **Provider** implements *ProviderInterface*

Fields

PATTERN

public static final String **PATTERN**

RANDOM_INT

public static final String **RANDOM_INT**

SEPARATOR

public static final String **SEPARATOR**

format

protected String **format**

populator

protected *PopulatorInterface* **populator**

Methods

getIdentifier

public String **getIdentifier**()

getPersonName

protected String **getPersonName**(String *type*)

getPopulator

public *PopulatorInterface* **getPopulator**()

mergeArrays

protected String[] **mergeArrays**(String[] *first*, String[] *second*)

parseFormat

protected String **parseFormat**(String *format*, *Callback* *callback*)

randomDouble

protected Double **randomDouble**()

randomDouble

protected Double **randomDouble**(int *minNumber*, int *maxNumber*)

randomElement

protected String **randomElement** (String[] *strings*)

randomElement

protected String **randomElement** (String[] *strings*, int *count*)

randomElements

protected String[] **randomElements** (String[] *strings*)

randomElements

protected String[] **randomElements** (String[] *strings*, int *count*)

randomInt

protected Integer **randomInt** ()

randomInt

protected Integer **randomInt** (int *minNumber*, int *maxNumber*)

setFormat

public *Provider* **setFormat** (String *format*)

setPopulator

public *ProviderInterface* **setPopulator** (*PopulatorInterface* *populator*)

toString

public String **toString** ()

ProviderInterface

public interface **ProviderInterface**
This class is responsible for generating data.

Methods

generate

String **generate**()

getIdentifier

String **getIdentifier**()

setPopulator

ProviderInterface **setPopulator**(*PopulatorInterface* *populator*)

RandomNumberProvider

public class **RandomNumberProvider** extends *Provider*

Fields

DECIMAL

public static final String **DECIMAL**

INTEGER

public static final String **INTEGER**

Constructors

RandomNumberProvider

public **RandomNumberProvider**()

Methods

generate

public String **generate**()

getDecimal

public *ProviderInterface* **getDecimal**()

setMax

public *RandomNumberProvider* **setMax**(int *min*)

setMin

public *RandomNumberProvider* **setMin**(int *max*)

Telephone

public class **Telephone** extends *Provider*

Constructors

Telephone

public **Telephone**()

Methods

generate

public String **generate**()

2.3.5 com.eddmash.form.fields

BaseField

public abstract class **BaseField**<V, E> implements *FieldInterface*<V, E>

Fields

form

protected *FormInterface* **form**

isEditable

protected boolean **isEditable**

Constructors

BaseField

BaseField(boolean *isEditable*)

Methods

getForm

public *FormInterface* **getForm**()

getValue

public abstract E **getValue**()

getView

public abstract V **getView**()

isEditable

public boolean **isEditable**()

setForm

public *FieldInterface* **setForm**(*FormInterface* *form*)

setValue

public abstract void **setValue**(E *o*)

CollectionField

public class **CollectionField** extends *BaseField*<List<View>, Object> implements *CollectionFieldInterface*<List<View>, Object>

Field that manipulates multiple individual views together.

Its important to note that the specific fields don't loose there individuality and the values return will be values for each single view.

Setting will be attempted on each single view if its value is found in the map of values passed in.

Constructors

CollectionField

public **CollectionField**(String *tag*)

CollectionField

public **CollectionField**(String *name*, boolean *isEditable*)

Methods

addField

public void **addField**(String *name*, View *view*)

addField

public void **addField**(*FieldInterface field*)

addField

public void **addField**(String *name*, View *view*, boolean *editable*)

getFields

public Map<String, *FieldInterface*> **getFields**()

getName

public String **getName**()

getValue

public Map<String, Object> **getValue**()

getView

public List<View> **getView**()

setValue

public void **setValue**(Object *o*)

CollectionFieldInterface

public interface **CollectionFieldInterface**<T, E> extends *FieldInterface*<T, E>

Field that manipulates multiple views together.

Its important to note that the specific fields don't loose there individuality and the values return will be values for each single view.

Setting will be attempted on each single view if its value is found in the map of values passed in.

Methods

addField

void **addField**(String *name*, View *view*)
Add view to the collection.

Parameters

- **name** – identify the view uniquely
- **view** – the view instance

addField

void **addField**(String *name*, View *view*, boolean *editable*)
Add view to the collection.

Parameters

- **name** – identify the view uniquely
- **view** – the view instance
- **editable** – indicate if view allows having its values being set, true if view is editable, else false.

addField

void **addField**(*FieldInterface field*)
Add field to the collection.

Parameters

- **field** – field to be added to the collection

getFields

Map<String, *FieldInterface*> **getFields**()
The fields that make up the collection

Returns map of fields. that are the in the collection field.

getValue

E **getValue**()

getView

T **getView**()

setValue

void **setValue**(E *o*)

FieldInterface

public interface **FieldInterface**<T, E> extends *FormAwareInterface*

This provides a consitent way of dealing with the different views provided by android.

Methods

getForm

FormInterface **getForm**()

The form instance this field is attached to.

Returns form

getName

String **getName**()

A name that uniquely identify the view. this is use when you need to pull a specific field from the form instance.

Returns name

getValue

E **getValue**()

Returns the value of the view

Throws

- ***FormException*** –

Returns Object

getView

T **getView**()

The actual view object(s) we are operating on.

Note this may return a list of view objects in case of CollectionField

Throws

- ***FormException*** – in case it not possible to retrieve the view object

Returns a view instance

isEditable

boolean **isEditable**()
Is the view editable, this tells the form not to set values for the view and also tells the populator not to populate it.

Returns true if editable, false otherwise

setValue

void **setValue**(E *o*)
Set view value.

Parameters

- **o** –

Throws

- ***FormException*** –

MultiField

public class **MultiField** extends *BaseField*<List<View>, Map> implements *MultiFieldInterface*
This fields deals with multi fields but they are each treated as one.

This mean the values return are for those fields that have values, the setting also happens to those fields who data is provided.

Constructors

MultiField

public **MultiField**(String *name*, boolean *isEditable*)

MultiField

public **MultiField**(String *name*)

Methods

addView

public void **addView**(String *id*, View *view*)

getChildCount

public int **getChildCount**()

getField

public *FieldInterface* **getField**(String *id*)

getFields

public List<*FieldInterface*> **getFields**()

getName

public String **getName**()

getValue

public Map **getValue**()

getView

public List<View> **getView**()

removeView

public void **removeView**(String *id*)

setValue

public void **setValue**(Map *o*)

MultiFieldInterface

public interface **MultiFieldInterface**

Methods

addView

void **addView**(String *id*, View *view*)

getChildCount

int **getChildCount**()
Get the number of views the multifield contains.

getField

FieldInterface **getField** (String *id*)

getFields

List<*FieldInterface*> **getFields** ()

removeView

void **removeView** (String *id*)

SimpleField

public class **SimpleField** extends *BaseField*<Object, Object>

Constructors

SimpleField

public **SimpleField** (String *name*, Object *value*)

SimpleField

public **SimpleField** (String *name*, Object *value*, boolean *isEditable*)

Methods

getName

public String **getName** ()

getValue

public Object **getValue** ()

getView

public Object **getView** ()

setValue

public void **setValue** (Object *o*)

ViewField

public class **ViewField** extends *BaseField*<View, String>

Constructors

ViewField

public **ViewField**(String *name*, View *view*)

ViewField

public **ViewField**(String *name*, View *view*, boolean *isEditable*)

Methods

getName

public String **getName**()

getSpinnerValuePosition

public int **getSpinnerValuePosition**(Spinner *spinner*, Object *val*)

getValue

public String **getValue**()

getView

public View **getView**()

setIsEditable

public void **setIsEditable**(boolean *isEditable*)

setValue

public void **setValue**(String *val*)

2.3.6 com.eddmash.form.values

Value

public class **Value** implements *ValueInterface*<String>

Constructors

Value

public **Value**(String *value*, String *label*)

Methods

getItem

public String **getItem**()

getLabel

public String **getLabel**()

getValue

public String **getValue**()

toString

public String **toString**()

ValueInterface

public interface **ValueInterface**<T>

Methods

getItem

T **getItem**()

getLabel

String **getLabel**()

getValue

String **getValue**()

ViewValue

public class **ViewValue** implements *ValueInterface*<Map>

Constructors

ViewValue

public **ViewValue** (Map *item*, String *labelCol*, String *valueCol*)

Methods

fromCollection

public static List<*ValueInterface*> **fromCollection** (List<Map> *data*, String *colKey*, String *valueKey*)

Take list of maps and prepares them for use as values on a spinner.

Parameters

- **data** –

getItem

public Map **getItem** ()

getLabel

public String **getLabel** ()

getValue

public String **getValue** ()

toString

public String **toString** ()

Index

I

L

M

N

P

R

S

T

V