VAADIN TRAINING

Forms

Overview

WHAT IS A FIELD - SOME BACKGROUND

- All UI components in Vaadin implement the Component interface
- Additionally we have a typed interface: **HasValue<V>**
- The **AbstractField** class implements both of these and is the basis for many of the field components in Vaadin

VALUE CONTAINER

HasValue is a typed interface that is implemented by components which have a bound data value that can be modified either from the code or from the UI via a field.

HANDLES VALUE CHANGES

You can register a **ValueChangeListener** to a field that implements **HasValue** and you will receive **ValueChangeEvents**.

OTHER FEATURES

- Read-only mode
- Required indicator visibility
- `isEmpty()`
- `clear()`
- `getEmptyValue()`

Vaadin Data Model

- Designed for modifying application data through UI components
- Works by binding Java Beans to Vaadin Components implementing HasValue<V>

BINDER

- Binder is the central and type safe piece dealing with bindings
- Binder takes care of the binding as well as Conversion and Validation
- Binder works directly with Java 8 method references

JAVA

```
Binder<Person> binder = new Binder<>();
binder.forField(titleField) // Start by defining the Field instance to use
  .bind(                    // Finalize by doing the actual binding to the Person
                            // class

    Person::getTitle,       // Callback that loads the title from a person instance
    Person::setTitle));     // Callback that saves the title in a person instance

binder.forField(titleField)
  .bind("title");           // Alternative bind method for Java beans
```

- Binder can also be used with Java 7 style ValueProvider and Setter instantiation.
- **@PropertyId** annotation can be used to name the components for automatically matching with bean property names
- Nested properties work through chained getters, no special support in Binder in framework 8.0

READING AND WRITING DATA

Unbuffered

JAVA

```
binder.setBean(person);  // Sets the person instance as a data source for the binder
```

Buffered

JAVA

```
binder.readBean(person);    // Reads values from the Person instance to the binder

try {
  binder.writeBean(person); // Writes values from the binder to the person interface
} catch (BindingException e) {
    // Could not save the values; check exceptions for each bound field
}
```


vaadin}>

VALIDATION

- Done as part of the binding through **withValidator**
- Supports defining Validator as Lambda function or Validator implementation
- The execution order is exactly as defined within the binding
- Validation works around **ValidationResult** and **ValueContext**.
- Some Validators are built in: **StringLengthValidator**, **RegExpValidator**, **EmailValidator**...
- **BinderValidationStatusHandler** can be implemented to customize how validation errors are visualized

JAVA

```
binder.forField(titleField)
  // Shorthand for requiring the field to be non-empty
  .asRequired("Every employee must have a title")
  .bind(Person::getTitle, Person::setTitle);
```

```
binder.forField(emailField)
  // Explicit validator instance
  .withValidator(new EmailValidator(“This doesn't look like a valid email address"))
  .bind(Person::getEmail, Person::setEmail);
```

```
binder.forField(nameField)
  // Validator defined based on a lambda and an error message
  .withValidator(
    name -> name.length() >= 3, "Full name must contain at least three characters")
  .bind(Person::getName, Person::setName);
```

CONVERSION

- Data type in Bean not always compatible with the data type of the Vaadin Component
- Converters are designed to convert between data types within the Binding
- Converters as well as Validators can be chained and their execution order will be exactly as specified.
- Basic converters are built in: **AbstractStringToNumberConverter**, **StringToDateConverter** etc...

JAVA

```
// Slider for integers between 1 and 10
Slider salaryLevelField = new Slider("Salary level", 1, 10);

binder.forField(salaryLevelField)
  .withConverter(Double::intValue, Integer::doubleValue)
  .bind(Person::getSalaryLevel, Person::setSalaryLevel);
```