

SHIP Validator - 0.3

Dag Hovland. Based on joint work with Khalid A. Mughal and Federico Mancini.

July 17, 2009

Premises

Annotations

Property-Tests

The Validation Process

Dynamic Typing

Handling null Values

Cross-Tests

Boolean Composition of Tests

Composed Property-Tests

Composed Cross-Tests

Error Messages

Related Work

Information as Properties

- ▶ Java is Object-Oriented — everything is an object
- ▶ Information available in no-arg methods: *properties*
- ▶ Input is represented as properties of an object

Goals

- ▶ A framework that should be easy to add onto existing projects
- ▶ It should be easy to use and create tests
- ▶ Tests should be reusable and composable
- ▶ Composition of tests should be easy and powerful
- ▶ No code inside methods in the class representing input
- ▶ No references to code by string names

Annotations

- ▶ From Java 5.0
- ▶ Standardizing comments
- ▶ A kind of interface
- ▶ On classes, methods, fields, variables

Scenario: International Money Transfer

IBAN

BIC

BICCODE

Account

Clearing-code

AB1232342

Amount

€ . c

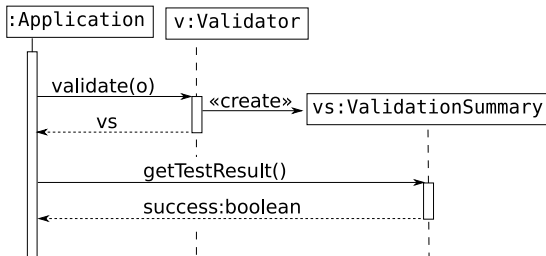
Configuration by Proximity

```
@ValidateBIC
public String getBIC() { return BIC; }

@IntRange(min=0,max=10000)
public Integer getAmountEuro() { return amountEuro; }

@IntRange(min=0,max=99)
public Integer getAmountCents() { return amountCents; }
```

Using the Framework




```
@Validation
public @interface IntRange {
    int min();
    int max();
    public static class Tester
    implements IPropertyTester<IntRange, Integer> {
        public boolean runTest(IntRange r, Integer v) {
            return(v >= r.min() && v <= r.max());
        }
    }
}
```

Interface for Property-Tests

```
public interface IPropertyTester <A extends Annotation,I>
{
    public boolean runTest(A an, I o)
        throws ValidationException;
}
```

Input Validation as Dynamic Typing

- ▶ `validate(o)` similar to down-casting in Java
- ▶ Why cannot Java's type hierarchy be used?
- ▶ `String` and `Integer` are `final`

@Required and @NotRequired

- ▶ But how should tests handle `null` values?
- ▶ They should, by default, lead to a crash
- ▶ However, `null` is also used to signal special cases
- ▶ `@Required` specifies that a property should not be `null` — almost standard
- ▶ `@NotRequired` causes no other tests to be run, if the value is `null`

Cross-Tests over Multiple Properties

- ▶ *Property-Tests* test a single property
- ▶ *Cross-Tests* test several properties at once

Scenario: International Money Transfer

IBAN

BIC

BICCODE

Account

Clearing-code

AB1232342

Amount

€ . c

Pay international bill

Using Cross-Tests

```
@ExactlyOneNull
@NotRequired
public String getIBAN() { return IBAN; }
```

```
@ExactlyOneNull
@AllOrNoneNull
@NotRequired
public String getAccount() { return account; }
```

```
@AllOrNoneNull
@NotRequired
public String getClearingcode() { return clearingcode; }
```

Declaring a Cross-Test

```
@CrossValidation
public @interface AllOrNonNull {
    public static class Tester
    implements ICrossTester<AllOrNonNull,String>{
        public boolean runTest(AllOrNonNull c,
                               ArrayList<String> v)
            throws ValidationException {
            ...
        }
    }
}
```



```
public interface ICrossTester <A extends Annotation, V> {  
    public boolean runTest(A a, ArrayList<V> v)  
        throws ValidationException;  
}
```

Basic and Composed Tests

- ▶ *Basic Tests* contain an inner class implementing a test
- ▶ *Composed Tests* are annotated with property-annotations

Property-Tests

```
@ValidateBIC  
public String getBIC() { return BIC; }
```

Example of Composed Property-Test

```
@Validation
@BoolTest(BoolType.AND)
@PatMatch("\\w8|\\w11")
@AdditionalTest
@interface ValidateBIC{}
```

Boolean Composition of Tests

- ▶ Annotation `@BoolTest`
- ▶ `public enum BoolType{OR, AND, ALLFALSE}`

Types of Tests

	Basic Tests	Composed Tests
Property-Tests	@IntRange	@ValidateBIC
Cross-Tests	@AllOrNonNull	@AmountCheck

Composed Cross-Tests

```
@InRange(min=0,max=10000)
@AmountCheck
public Integer getAmountEuro() { return amountEuro; }
```

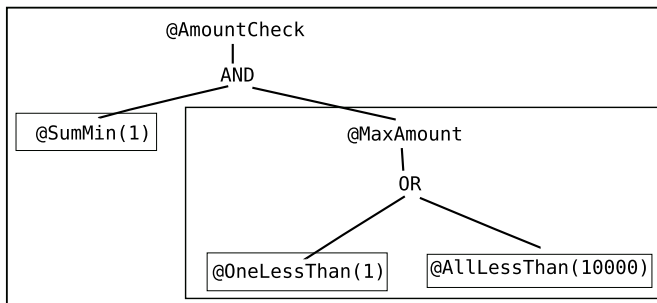
```
@InRange(min=0,max=99)
@AmountCheck
public Integer getAmountCents() { return amountCents; }
```

Declaring Composed Cross-Tests

```
// MaxAmount declaration
@CrossValidation
@BoolTest(BoolType.OR)
@OneLessThan(1)
@AllLessThan(10000)
public @interface MaxAmount {}
```

```
//AmountCheck declaration
@CrossValidation
@BoolTest(BoolType.AND)
@SumMin(1)
@MaxAmount
public @interface AmountCheck {}
```


@AmountCheck



Custom Error Message

```
@Validation
@BoolTest(BoolType.AND)
@PatMatch("\\w8|\\w11")
@AdditionalTest
@ValErr(message="Invalid BIC")
public @interface ValidateBIC{}
```

The value "BICCODE" returned by method "getBIC" has not passed the following property-test:

```
+Test: @ValidateBIC()  
  (Error Message: Invalid BIC) because:  
  | -+Test: @PatMatch(value=\w8|\w11)
```

=====
The following cross-tests have failed:

```
+Test: @AmountCheck() because:  
  | -+Test: @MaxAmount() because:  
    | -+Test: @OneLessThan(value=1)  
    | -+Test: @AllLessThan(value=10000)  
+Test: @AllOrNonnull()  
+Test: @ExactlyOneNull() because:  
  | -+Test: @ExactlyNNull(value=1)
```

- ▶ Heimdall
- ▶ Struts 2
- ▶ Java Specification Request (JSR) 303: Bean Validation
- ▶ Hibernate
- ▶ Hookom and Holmgren

`http://www.owasp.org/index.php/Category:
OWASP_Content_Validation_using_Java_Annotations_
Project_-_SHIP_Validator_0.3_Release`