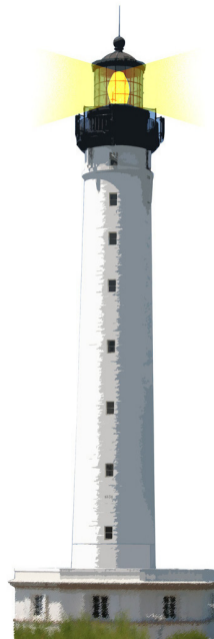


# Parametrized Tests

Getting more tests out of test cases

S. Ducasse



# Goal

- How to **reuse** test logic with specific value sets?
- How can we run tests with **all** possible combinations?
- Use a naive example to stress the essence



# Problem one

```
MyDullTest >> testSum  
self assert: (2/3) + (1/3) equals: 1
```

- How to generalize it?
- How can we reuse test logic with specific values?



# Using a collection

```
MyDullTest >> testSum
```

```
{ { 2 . 1 . 3 } . { 2/3 . 1/3 . 1 } }  
do: [ :each |  
  self  
  assert: each first + each second  
  equals: each third ]
```

- What if I want to have **another** test **reusing** some or all these values?
- Not nice to have to duplicate the loop.



# Use a parametrized test

Inherit from `ParametrizedTestCase`, add instance variables, and setters.

```
ParametrizedTestCase < #MyDullTest  
  slots: { #number1 . #number2 . #result}
```

Use instance variables:

```
MyDullTest >> testSum  
  self assert: number1 + number2 equals: result
```



# Declare cases

MyTest class >> testParameters

```
^ ParametrizedTestMatrix new
  addCase: { #number1 -> 2. #number2 -> 1.0. #result -> 3 };
  addCase: { #number1 -> (2/3). #number2 -> (1/3). #result -> 1 };
  yourself
```



# Benefits

We can add new tests that use the variables

```
MyDullTest >> testSum  
self.assert: result - number2 equals: number1
```

We can execute tests

- with a specific configuration
- add a new configuration



## Problem two

Imagine we have a test and we would like to apply to other collections.

```
MyTest >> testAdd
```

```
| aCollection |
```

```
aCollection := Bag new.
```

```
aCollection add: 'a'.
```

```
self assert: (aCollection includes: 'a').
```

```
self assert: aCollection size equals: 1.
```



Introduce a setter for the class and use it.

```
MyTest >> testAdd
```

```
| aCollection |
```

```
aCollection := collectionClass new.
```

```
aCollection add: 'a'.
```

```
self assert: (aCollection includes: 'a').
```

```
self assert: aCollection size equals: 1.
```



# Inherit from ParametrizedTestCase

```
ParametrizedTestCase << #MyTest  
  slots: {#collectionClass};  
  package: 'MyTests'
```

# Declare test parameters

```
MyTest class >> testParameters
```

```
  ^ ParametrizedTestMatrix new  
  forSelector: #collectionClass  
  addOptions: { Set . Bag . OrderedCollection }
```

We run all the tests with Set, Bag, and OrderedCollection.



# We want more

We would like to have **different items** to add and to check with all the collections.



Introduce an instance variable and setter for one item and use it.

```
ParametrizedTestCase << #MyTest  
  slots: { #collection . #item };  
  package: 'MyTests'
```

```
MyTest >> testAdd
```

```
| aCollection |  
aCollection := collection class new.  
aCollection add: item.  
self assert: (aCollection includes: item).  
self assert: aCollection size equals: 1.
```

# Declare test parameters

MyTest class >> testParameters

```
^ ParametrizedTestMatrix new
  forSelector: #item addOptions: { 1 . 'a' . $c };
  forSelector: #collectionClass addOptions: { Set . Bag . OrderedCollection }
```

- collectionClass and item will take the values from the options
- Tests with all possible combinations of item and collectionClass are then run



# Conclusion

- Parametrized tests are handy
- You can get more tests out of your testcase



A course by

S. Ducasse, G. Polito, and Pablo Tesone



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France  
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>