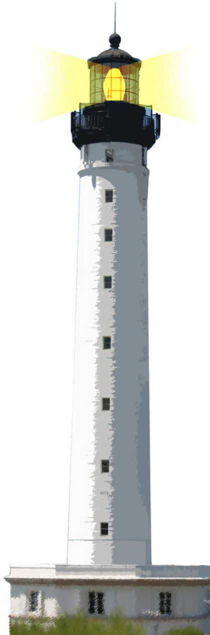
Avoid hardcoding classes

S. Ducasse



http://www.pharo.org

Goal

- Think that a class is a kind of global
- Think about parametrization

A simple case

```
EFTest >> testAssignment

  | source expr |
  expr := RBParser parseExpression: 'a:=1'.
  source := (EFFormatter new installNewContext:
          (self perform: configurationSelector) yourself) format: expr.
  self assert: source equals: 'a := 1'
```

What if we want to check that an alternate Formatter is satisfying the test?

Solution

- Do not hardcode class name
- Define test parameters

Step 1

```
EFTest >> testAssignment

  | source expr |
  expr := self parserClass parseExpression: 'a:=1'.
  source := (self formatterClass new installNewContext:
          (self perform: configurationSelector) yourself) format: expr.
  self assert: source equals: 'a := 1'
```

```
EFTest >> parserClass
  ^ RBParser
```

```
EFTest >> formatterClass
  ^ EFFormatter
```

Step 2: state and setter

```
EFTest >> formatterClass: aFormatterClass
  formatterClass := aFormatterClass
```

Step 3: introducing test parameters

```
EFTest class >> testParameters

  ^ ParametrizedTestMatrix new
    addCase: { #formatterClass —> EFFormatter. #contextClass —> EFContext };
    addCase: { #formatterClass —> AlternateFormatter. #contextClass —> EFContext };
    yourself.
```

- All the tests will run for each configuration.
- Now we can turn parserClass as a test parameter if needed!

Having a nice logic

```
EFTest >> testAssignment

  | source expr |
  expr := self parserClass parseExpression: 'a:=1'.
  source := (self formatterClass new installNewContext:
          (self perform: configurationSelector) yourself) format: expr.
  self assert: source equals: 'a := 1'
```

into

```
testAssignment
  | source expr |
  expr := self parseExpression: 'a:=1'.
  source := self formatter format: expr.
  self assert: source equals: 'a := 1'
```

And finally

```
EFTest >> testAssignment
  self
    assert: (self formatExpression: 'a:=1')
    equals: 'a := 1'
```

Conclusion

- Factor out class references
- Ease extension by overriding (Remember self-sends are plans for reuse)
- Support test parametrization

A course by

S. Ducasse, L. Fabresse, G. Polito, and Pablo Tesone