Pillar Booktester

RMod

Table of contents

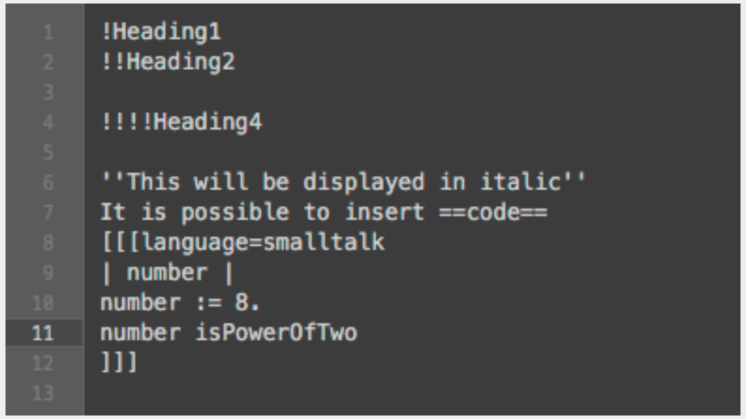
What is Pillar?

Markup syntax
Associated tools

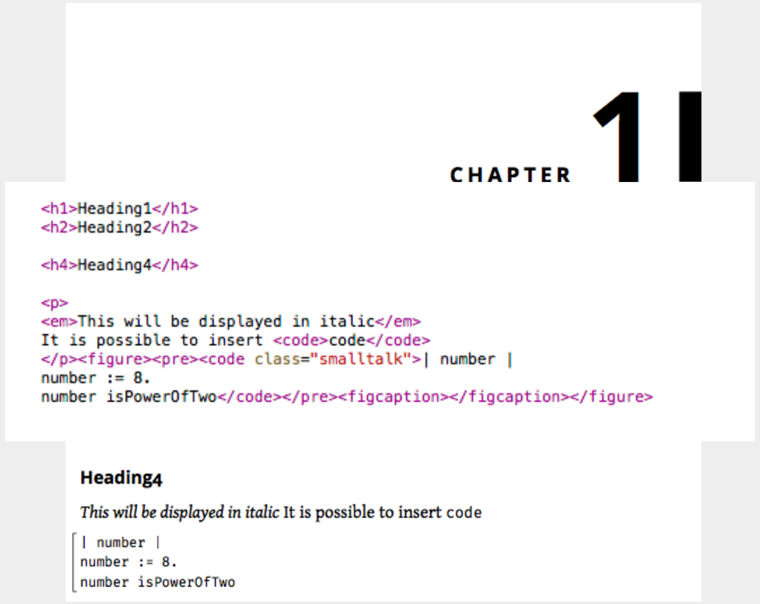
Common syntax

Documentation

Different outputs and uses from a single syntax



```
!Heading1
!!Heading2

!!!!Heading4

''This will be displayed in italic''
It is possible to insert ==code==
[[[language=smalltalk
| number |
number := 8.
number isPowerOfTwo
]]]
```



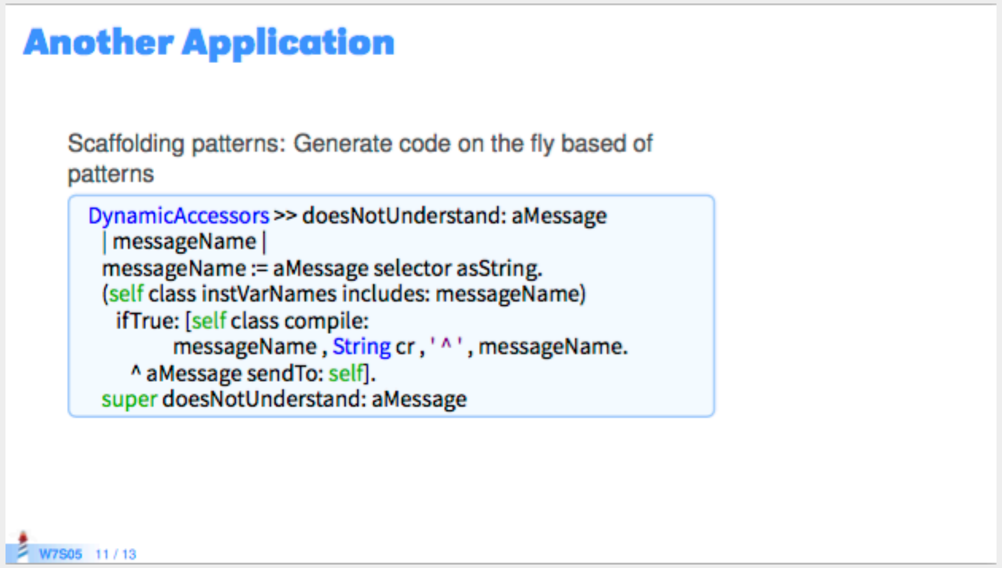
CHAPTER 11

```
<h1>Heading1</h1>
<h2>Heading2</h2>

<h4>Heading4</h4>

<p>
<em>This will be displayed in italic</em>
It is possible to insert <code>code</code>
</p><figure><pre><code class="smalltalk">| number |
number := 8.
number isPowerOfTwo</code></pre><figcaption></figcaption></figure>
```

Heading4

This will be displayed in italic It is possible to insert `code`

```
| number |
number := 8.
number isPowerOfTwo
```



Another Application

Scaffolding patterns: Generate code on the fly based of patterns

```
DynamicAccessors >> doesNotUnderstand: aMessage
  | messageName |
  messageName := aMessage selector asString.
  (self class instVarNames includes: messageName)
    ifTrue: [self class compile:
          messageName , String cr , ' ^ ' , messageName.
      ^ aMessage sendTo: self].
  super doesNotUnderstand: aMessage
```

W7S05 11 / 13

Testing a book?

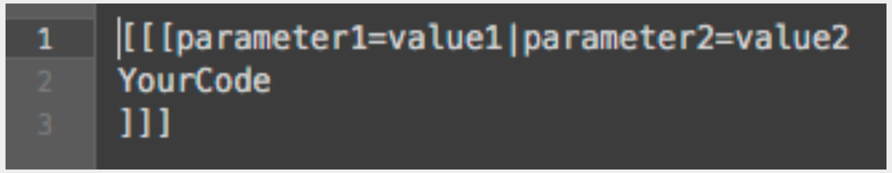
What does it mean?

- Test the code it displays

Why is it useful?

- Updating is easier
- Updating becomes more frequent

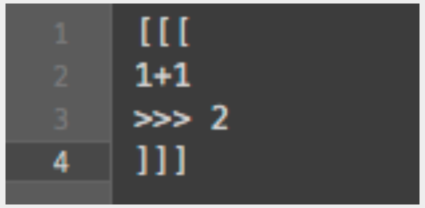
Codeblocks → Display your code



```
[[[parameter1=value1|parameter2=value2
YourCode
]]]
```

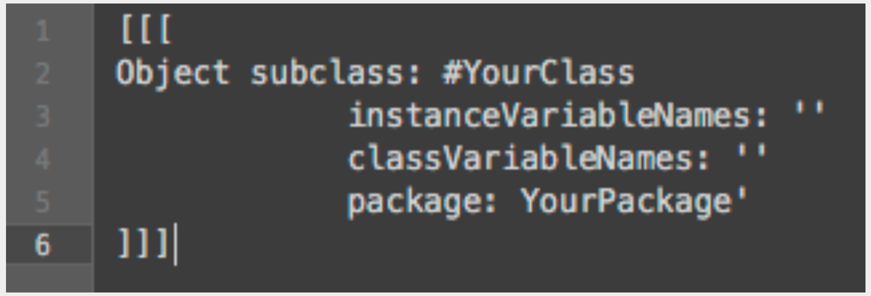
3 types

Examples



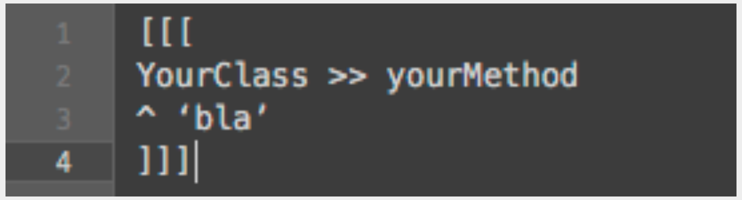
```
[[[
1+1
>>> 2
]]]
```

Class definitions



```
[[[
Object subclass: #YourClass
          instanceVariableNames: ''
          classVariableNames: ''
          package: YourPackage'
]]]
```

Method definitions



```
[[[
YourClass >> yourMethod
^ 'bla'
]]]
```

Testing codeblocks

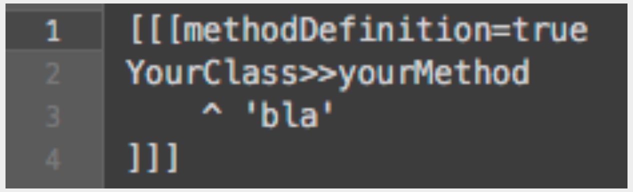
Making your codeblocks testable

- classDefinition
- methodDefinition
- example

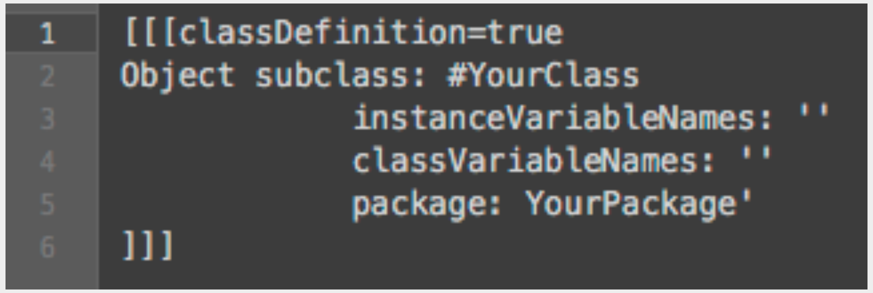
Visitor

- Visits codeblocks
- Collects PRResult

Method and class definitions



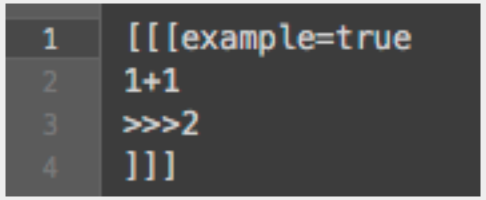
```
[[[methodDefinition=true
YourClass>>yourMethod
    ^ 'bla'
]]]
```



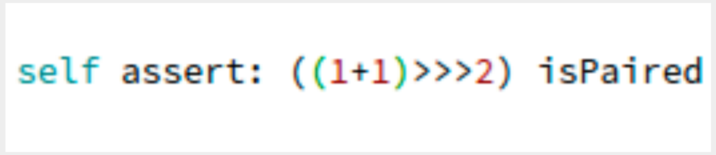
```
[[[classDefinition=true
Object subclass: #YourClass
            instanceVariableNames: ''
            classVariableNames: ''
            package: YourPackage'
]]]
```

Correct compilation

Examples



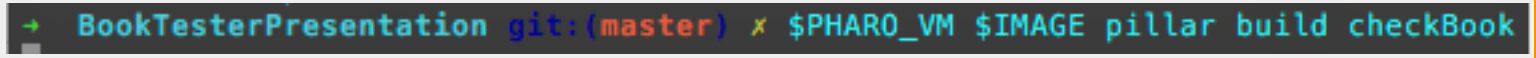
```
[[[example=true
1+1
>>>2
]]]
```



```
self assert: ((1+1)>>>2) isPaired
```

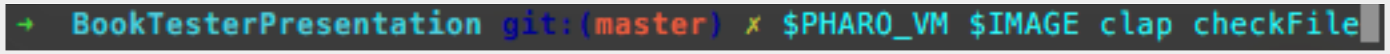
Command line & Demo

Two different command lines



```
➜  BookTesterPresentation git:(master) ✗ $PHARO_VM $IMAGE pillar build checkBook
```

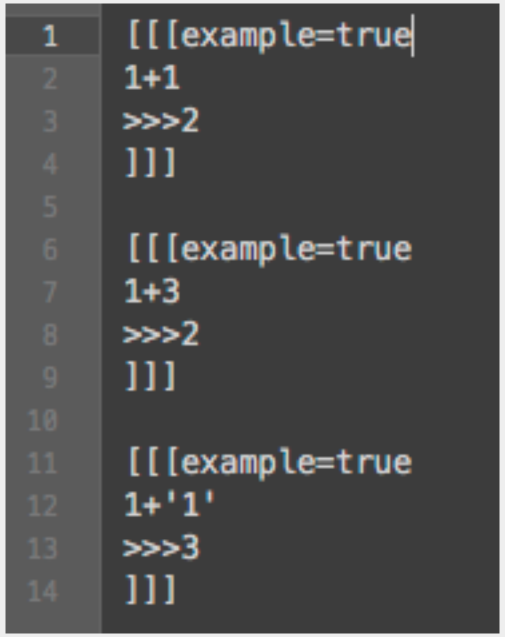
Tests all the .pillar files recursively in the directory



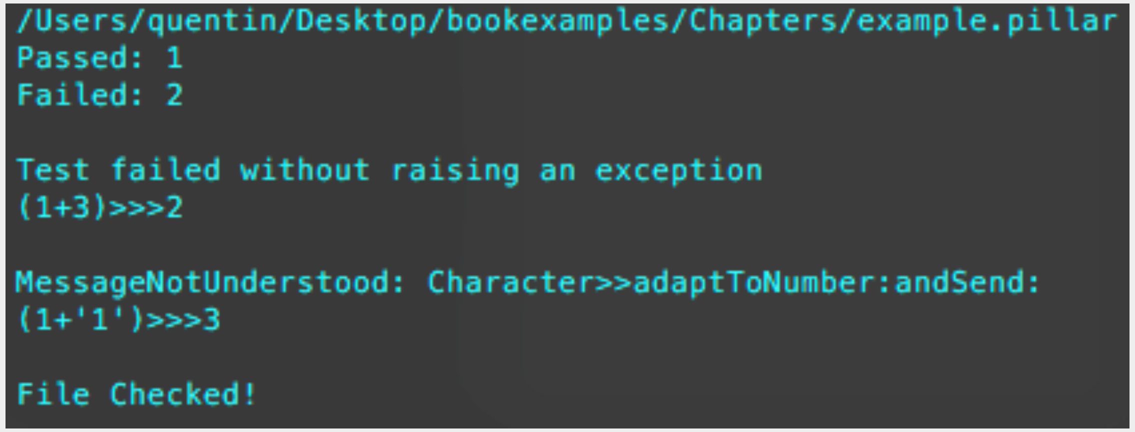
```
➜  BookTesterPresentation git:(master) ✗ $PHARO_VM $IMAGE clap checkFile
```

Tests the .pillar file in the directory

Rendering



```
[[[example=true
1+1
>>>2
]]]

[[[example=true
1+3
>>>2
]]]

[[[example=true
1+'1'
>>>3
]]]
```

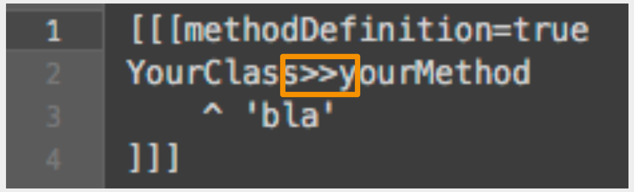


```
/Users/quentin/Desktop/bookexamples/Chapters/example.pillar
Passed: 1
Failed: 2

Test failed without raising an exception
(1+3)>>>2

MessageNotUnderstood: Character>>adaptToNumber:andSend:
(1+'1')>>>3

File Checked!
```

Limitations

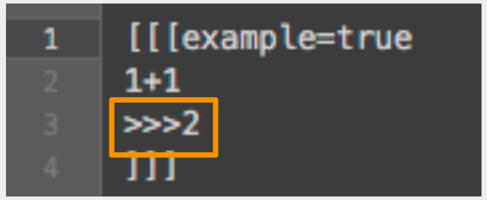
Rigid syntax

>>> for examples
>> for method definitions



```
[[[methodDefinition=true
YourClass>>yourMethod
    ^ 'bla'
]]]
```

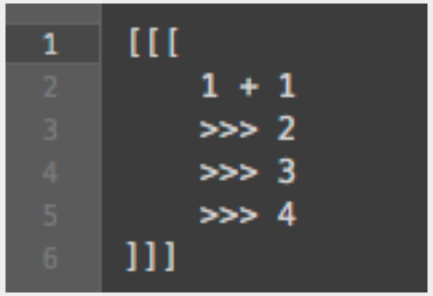
The class has to be defined before



```
[[[example=true
1+1
>>>2
]]]
```

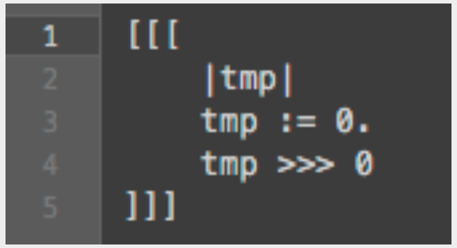
Unaccepted codeblock types

- Iterations



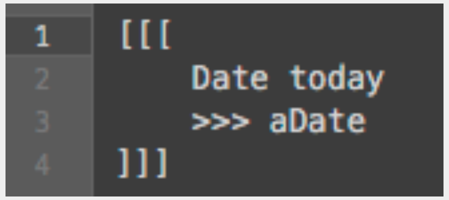
```
[[[
    1 + 1
    >>> 2
    >>> 3
    >>> 4
]]]
```

- Local variables



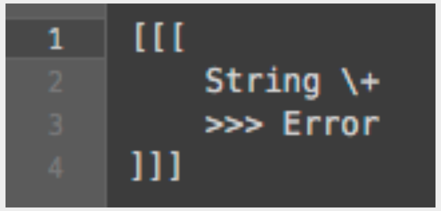
```
[[[
    |tmp|
    tmp := 0.
    tmp >>> 0
]]]
```

- Instanciation



```
[[[
    Date today
    >>> aDate
]]]
```

- Exception raising



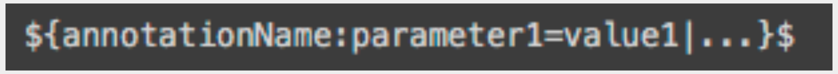
```
[[[
    String \+
    >>> Error
]]]
```

Improving Book Writability

New annotations

- showClass
- showMethod
- screenshot
- loader
- run

Note: This is how an annotation works

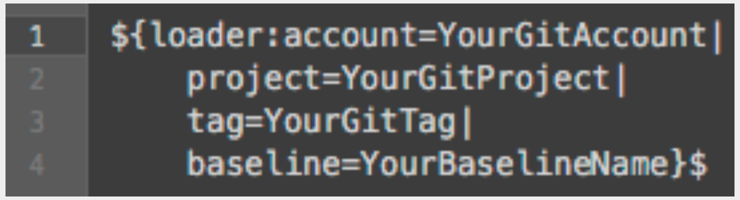


```
${annotationName:parameter1=value1|...}$
```

Transformed or not

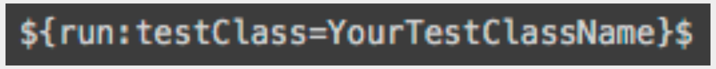
column **loader**

loader



```
${loader:account=YourGitAccount|
    project=YourGitProject|
    tag=YourGitTag|
    baseline=YourBaselineName}$
```

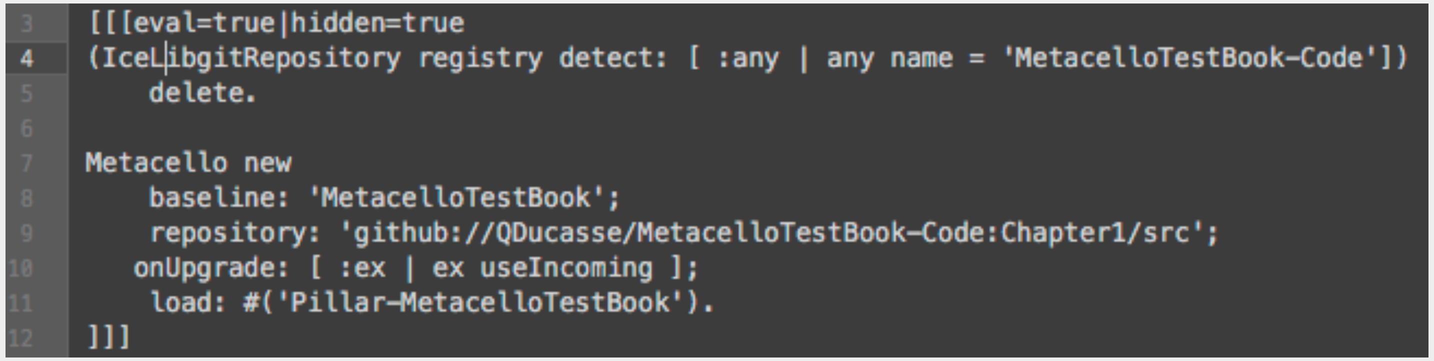
run



```
${run:testClass=YourTestClassName}$
```

Runs every test in the given class

No transformers!
New way of executing code



```
[[[eval=true|hidden=true
(IceLibgitRepository registry detect: [ :any | any name = 'MetacelloTestBook-Code'])
    delete.

Metacello new
    baseline: 'MetacelloTestBook';
    repository: 'github://QDucasse/MetacelloTestBook-Code:Chapter1/src';
   onUpgrade: [ :ex | ex useIncoming ];
    load: #('Pillar-MetacelloTestBook').
]]]
```

New annotations

showMethod

```
${showMethod:method=isPowerOfTwo|class=Integer}$
```

```
[[[
Integer>>isPowerOfTwo
    "Return true if the receiver is an integral power of two."
    ^ self ~= 0 and: [(self bitAnd: self-1) = 0]
]]]
```

showClass

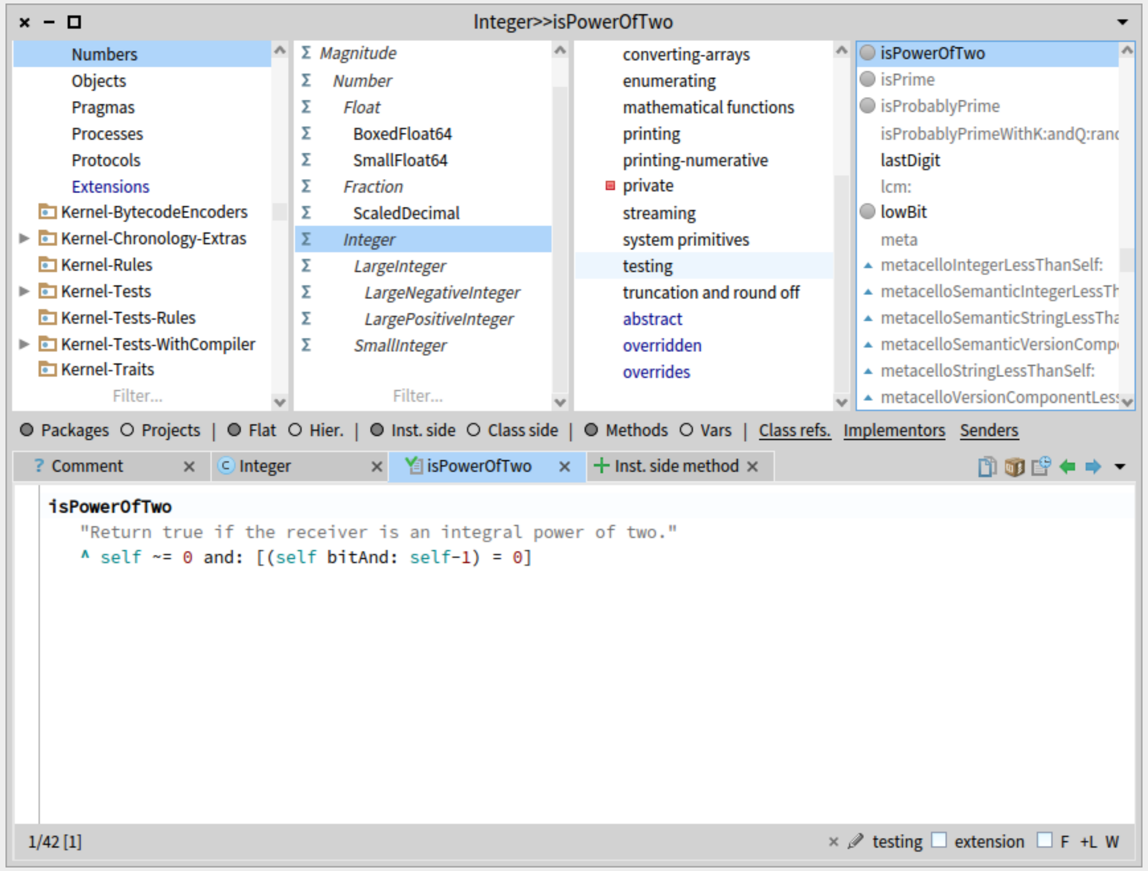
```
${showClass:class=Integer}$
```

```
[[[
Number subclass: #Integer
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'Kernel-Numbers'
]]]
```

screenshot

```
${screenshot:class=YourClassName|
             method=yourMethodName|
             caption=yourCaption|
             width=yourWidth|
             label=yourLabel}$
```

```
+yourCaption.>picturePath|width=yourWidth
                         |label=yourLabel+
```



```
isPowerOfTwo
    "Return true if the receiver is an integral power of two."
    ^ self ~= 0 and: [(self bitAnd: self-1) = 0]
```

Demo 2

Thank you!