



## Object-Oriented Design Lecture

# Xtreme Test Driven

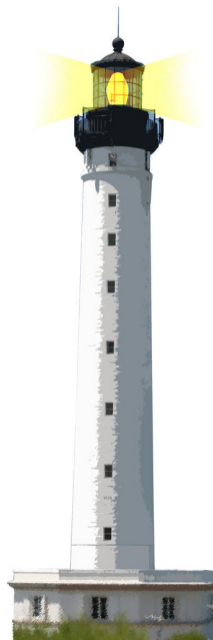
S. Ducasse

<http://stephane.ducasse.free.fr>

<http://car.mines-douai.fr/luc>



<http://www.pharo.org>



# Outline

- TDD on steroids
- Live programming
- Smart tools
- Absolutely gorgeous development flow



# Principle

- Do not break the flow
- Write a test
- When it breaks, compile method on the fly in the debugger
- Resume and continue until test is green



# Studying an example

- A dead simple counter. Nothing simpler.
- Focus on essence of the process!
- You can do it.



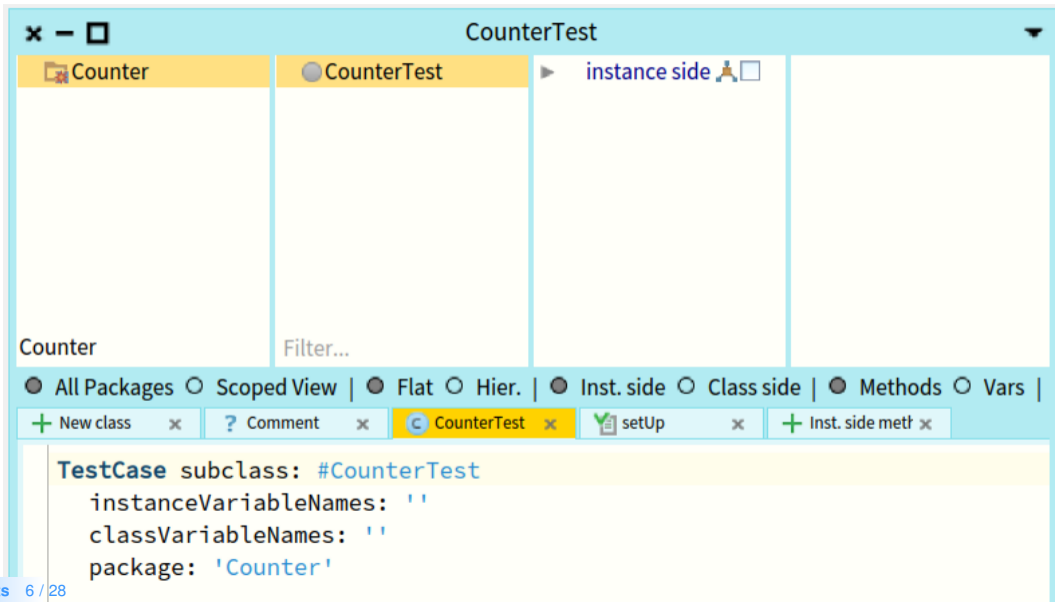
# An Empty Package

The screenshot shows an IDE window titled "Counter". The top-left pane displays a file explorer with a folder named "Counter". The bottom pane is a code editor showing a new class template:

```
Object subclass: #NameOfSubclass
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Counter'
```

Below the code editor, there is a navigation bar with the following options:  All Packages,  Scoped View,  Inst. side, and  Class side. A toolbar below this bar includes a help icon, a comment icon, a "New class" button, and navigation arrows.

# An Empty Test Case Class



The screenshot shows an IDE window titled "CounterTest". The interface is divided into several panes. On the left, there is a "Counter" class icon. The main area is split into two columns: "CounterTest" (selected) and "instance side". Below the main area, there is a toolbar with various view options: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side" (selected), "Class side", "Methods", and "Vars". Below the toolbar, there is a tab bar with "CounterTest" selected. The main editor area displays the following code:

```
TestCase subclass: #CounterTest
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Counter'
```

At the bottom left, there is a small icon of a lighthouse and the text "Tests 6 / 28".

# A first test

The screenshot shows an IDE window titled "CounterTest". The interface is divided into several panes:

- Top Left:** A tree view showing the project structure with "Counter" and "CounterTest" folders.
- Top Middle:** A search filter input field labeled "Filter...".
- Top Right:** A pane showing the current view, labeled "instance side".
- Bottom Left:** A toolbar with radio buttons for "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", and "Vars".
- Bottom Middle:** A tab bar with tabs for "Comment", "CounterTest", "setUp", and "\*Inst. side met".
- Bottom Right:** A code editor showing the test method `testSetAndGetCounter` with the following code:

```
self assert: (Counter new count: 22) count equals: 22
```

# A first test

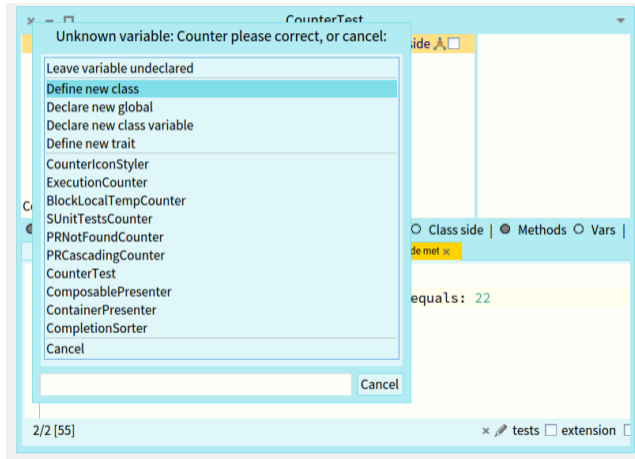
```
testSetAndGetCounter
self assert: (Counter new count: 22) count equals: 22
```

- Method is about to be compiled
- The system knows the class does not exist!



# Define a class

- At compile time...



# Define a class (II)

The screenshot shows an IDE window titled "CounterTest" with a breadcrumb path: Counter > CounterTest > instance side. A dialog box titled "Information Required" is open, prompting to "Edit class definition:". The dialog contains a text area with the following code:

```
Object subclass: #Counter  
  instanceVariableNames: ""  
  classVariableNames: ""  
  category: 'Counter'
```

At the bottom of the dialog are "OK" and "Cancel" buttons. In the background, the IDE interface shows a "Counter" class with a "testSetAndGet" method and a "self assert" statement.

# Test Defined but Not Executed

The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Project Explorer:** Shows a package "Counter" containing a class "CounterTest".
- Class Explorer:** Shows the "Counter" class with a red exclamation mark icon, indicating an error or issue.
- Method Explorer:** Shows the "tests" package containing the method "testSetAndGetCounter".
- Code Editor:** Displays the source code for the "testSetAndGetCounter" method. The code is as follows:

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

At the bottom of the IDE, there is a toolbar with various view options: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", and "Vars". Below the toolbar, there are tabs for "Comment", "CounterTest", "setUp", "testSetAndGet", and "Inst. side metr".

# Running the test

The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** A tree view showing the package structure. "Counter" is selected.
- Middle-Left Pane:** A list of classes. "Counter !" and "CounterTest" are visible. "CounterTest" is selected.
- Middle-Right Pane:** A list of test methods. "instance side" and "tests" are visible. "tests" is selected.
- Right Pane:** A list of test cases. "testSetAndGetCounter" is visible and selected.

Below the panes, there is a toolbar with the following options:  All Packages,  Scoped View,  Flat,  Hier.,  Inst. side,  Class side,  Methods,  Vars, and a search icon.

At the bottom, there is a code editor showing the following code:

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

At the bottom left of the slide, there is a small icon of a lighthouse and the text "Tests 12 / 28".

# First Error

Instance of Counter did not understand #count: Bytecode GT

Stack + Create ▶ Proceed ↺ Restart ↻ Step into ↗ Step over ↘ Step through -

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core
FullBlockClosure(BlockClosure)	ensure:		Kernel

Source Where is? Browse

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

Variables Evaluator

Type	Variable	Value
implicit	self	CounterTest>>#testSetAndGetCounter
attribute	expectedFails	an Array [0 items] ()
attribute	testSelector	#testSetAndGetCounter
implicit	thisContext	CounterTest>>testSetAndGetCounter

# Create a method on the fly

Create the missing class or method in the user prompted class, and restart the debugger at the location where it can be edited.

Instance of Counter d Bytecode GT ▾

Stack + Create ▶ Proceed ↻ Restart ⚙ Step into ⏪ Step over ⏩ Step through ☰

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core
FullBlockClosure(BlockClosure)	ensure:		Kernel

Source 🔍 Where is? 📄 Browse

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

# Create a method on the fly (II)

Instance of Counter did not understand #count: Bytecode GT

Stack ▶ Proceed ◀ Restart ▶ Step into ▶ Step over ▶ Step through ▾

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source 🔍 Where is? 📄 Browse

```
count: anInteger  
self shouldBeImplemented.
```

Variables Evaluator

Type	Variable	Value
implicit	self	a Counter

# Edit the method in the debugger (III)

The screenshot shows an IDE debugger window titled "Instance of Counter did not understand #count:". The window is divided into several sections:

- Stack:** A table showing the current stack frame. The selected frame is for the `count:` method in the `Counter` class.
- Source:** A code editor showing the source code for the `count:` method. The current line is `count := anInteger`.
- Variables:** A table showing the current state of variables.

**Stack Trace:**

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

**Source Code:**

```
count: anInteger
  count := anInteger
```

**Variables Table:**

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22
implicit	thisContext	Counter>>count:
implicit	stack top	22



# Add an instance variable on the fly

The screenshot shows an IDE window titled "Instance of Counter did not understand #count:". A modal dialog box is open with the message "Unknown variable: count please correct, or cancel:". The dialog has three options: "Declare new temporary variable", "Declare new instance variable" (which is highlighted), and "Cancel".

Below the dialog, the "Source" editor shows the following code:

```
count: anInteger  
  count := anInteger
```

The "Variables" panel at the bottom shows the current state of the program:

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22
implicit	thisContext	Counter>>count:
implicit	stack top	22

# Compile....

Instance of Counter did not understand #count: Bytecode GT

Stack ▶ Proceed ◀ Restart ↩ Step into ↗ Step over ↘ Step through ≡

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source 🔍 Where is? 📄 Browse

```
count: anInteger  
  count := anInteger|
```

# Continue the execution...

Instance of Counter did not un...  
Relinquish debugger control and proceed execution from the current point of debugger control.cmd+r

Bytecode GT ▾

Stack ▶ Proceed ◀ Restart ↩ Step into ↪ Step over ↪ Step through ≡

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source 🔍 Where is? 📄 Browse

```
count: anInteger  
count := anInteger|
```

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22

# Stepping back

- The system created a new method
- Removed the stack element with Error
- Replace it with a call to the new method
- Relaunches execution
- We edited it and recompiled
- Continued execution



## Stepping back (II)

- The system created a new method
- Removed the stack element with Error
- Replace it with a **call** to the new method

```
count: anInteger  
self shouldBeImplemented
```

- `shouldBeImplemented` is just an exception so that the debugger stops again



# Same story....

Instance of Counter did not understand #count Bytecode GT

Stack + Create ▶ Proceed ↺ Restart ↻ Step into Step over Step through -≡

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Cor
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Cor
FullBlockClosure(BlockClosure)	ensure:		Kernel

Source Where is? Browse

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

# Smart tools precompile methods

Instance of Counter did not understand #count

Bytecode GT

Stack

Proceed Restart Step into Step over Step through

Class	Method	Other	Package
Counter	count		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Cor
CounterTest(TestCase)	runCase	[self setUp. self performTest SUnit-Cor	

Source

Where is? Browse

```
count
^ count
```

Variables

Type	Variable	Value
implicit	self	a Counter
attribute	count	22
implicit	thisContext	Counter>>count
implicit	stack top	nil

# Test is green

The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** A tree view showing the package structure. "Counter" is selected, and "CounterTest" is highlighted under it.
- Middle Pane:** A view of the selected class, showing "instance side" and "tests".
- Right Pane:** A view of the selected test method, "testSetAndGetCounter", which is marked with a green circle, indicating it passed.

Below the panes is a toolbar with various view options: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", and "Vars".

At the bottom, a code editor shows the following code snippet:

```
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

The status bar at the bottom left indicates "Tests 24 / 28".



# One Cycle

- Run all the tests
- Ready to commit
- New test



# Stepping back

- Avoid guessing when coding
- Much much better context
  - inspect that specific instance state
  - talk to that specific object
- Inspectable / interactable context
- Tests are not a side effect artefacts but the driving force



# Protip from expert Pharo developers

- Get as fast as possible one object
- Cristalize your scenario with a test
- Xtreme TDD
- Loop



A course by

Stéphane Ducasse

<http://stephane.ducasse.free.fr>

and

Luc Fabresse

<http://car.mines-douai.fr/luc>



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/>