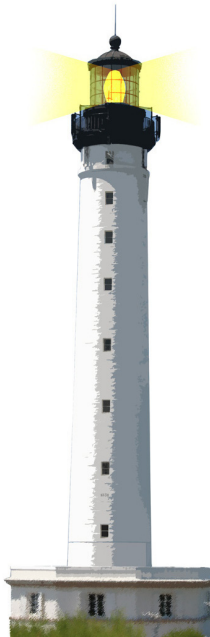
Learning Object-Oriented Programming and Design with TDD

About Instance Initialization

Stéphane Ducasse
http://stephane.ducasse.free.fr



Pharo
http://www.pharo.org

How to ensure that an instance is well initialized?

Possible solution

- Automatic initialize
- Lazy initialization
- Proposing the right interface
- Providing a default value

Provider responsibility

- This is the responsibility of the class to provide well-formed objects
- A client should not make assumptions or been responsible to send specific sequence of messages to get a working object

A First Implementation of Packet

```
Object subclass: #Packet
  instanceVariableNames: 'contents addressee originator '

Packet >> printOn: aStream
  super printOn: aStream.
  aStream nextPutAll: ' addressed to: '; nextPutAll: self addressee.
  aStream nextPutAll: ' with contents: '; nextPutAll: self contents

Packet >> addressee
  ^ addressee
Packet >> addressee: aSymbol
  addressee := aSymbol
```

Example of instance creation

```
Packet new
  addressee: #mac ;
  contents: 'hello mac'
```

Fragile Instance Creation

If we do not specify a contents, it breaks!

```
| p |
p := Packet new addressee: #mac.
p printOn: aStream
–> error
```

Problems

- Responsibility of the instance creation relies on the clients
- A client can create packet without contents, without address and the instance variables are not initialized correctly
- error (for example, printOn:)
- Fragile system (printOn: should be robust)

Fragile Instance Creation Solutions

- Automatic initialization of instance variables
- Proposing a solid interface for the creation
- Lazy initialization
- Providing a default value

Assuring Instance Variable Initialization

- How to initialize a newly created instance?
- Define the method initialize

```
Packet >> initialize
  super initialize.
  contents := ''.
  addressee := #noAd
```

- Makes sure that contents and addressee have always a value

The New/Initialize Couple

```
Object >> initialize
  "do nothing. Called by new my subclasses override me if necessary"
  ^ self
```

- Acts as a constructor in other languages

Lazy Initialization

- When some instance variables are:
  - not used all the time
  - consuming space, difficult to initialize because depending on other
  - need a lot of computation
- Use lazy initialization based on accessors
- But accessor access should be used consistently!

Lazy Initialization Example

A lazy initialization scheme with default value

```
Packet >> contents
  contents isNil
    ifTrue: [ contents := 'no contents' ]
  ^ contents
```

- aPacket contents or self contents
- A lazy initialization scheme with computed value

```
Dummy >> ratio
  ratio isNil
    ifTrue: [ ratio := self heavyComputation ]
```

Better

```
Packet >> contents
  contents isNil
    ifTrue: [contents := 'no contents']
  ^ contents
```

is equivalent to

```
Packet >> contents
  ^ contents ifNil: [ contents := 'no contents ]
```

Strengthen instance creation interface

- Problem: A client can still create aPacket without address.
- Solution: Force the client to use the class interface creation.
- Providing an interface for creation and avoiding the use of new: Packet send: 'Hello mac' to: #Mac

```
Packet class >> send: aString to: anAddress
  ^ self new contents: aString ; addressee: anAddress ; yourself
```

Examples of instance initialization

- Step 1. SortedCollection sortBlock: [:a :b| a name < b name]

```
SortedCollection class>>sortBlock: aBlock
  "Answer a new instance of SortedCollection such that its elements are sorted
     according to the criterion specified in aBlock."
  ^ self new sortBlock: aBlock
```

- Step 2. self new => aSortedCollection
- Step 3. aSortedCollection sortBlock: aBlock

Another example

- Step 1. OrderedCollection with: 1

```
Collection class >> with: anObject
  "Answer a new instance of a Collection containing anObject."
  | newCollection |
  newCollection := self new.
  newCollection add: anObject.
  ^ newCollection
```

Invoking per default the creation interface

```
OrderedCollection class >> new
  "Answer a new empty instance of OrderedCollection."
  ^ self new: 5
```

Forbidding new?

- Problem: We can still use new to create fragile instances
- Solution: new should raise an error!

```
Packet class >> new
  self error: 'Packet should only be created using send:to:'
```

Forbidding new implications

But we still have to be able to create instance!

```
Packet class >> send: aString to: anAddres
  ^ self new contents: aString ; addressee: anAddress
```

raises an error

```
Packet class >> send: aString to: anAddress
  ^ super new contents: aString ; addressee: anAddress
```

Forbidding new

Solution: use basicNew and basicNew:

```
Packet class >> send: aString to: anAddress
  ^ self basicNew
    contents: aString ;
    addressee: anAddress
```

Conclusion: Do not override basic* methods else you will not be able to invoke them later

Fluid vs. and default creation interface

Often a fixed creation interface gets too large

- Too many possibilities
- Too many optional cases

Example: Problem with class definition in Pharo

```
Object subclass: #Behavior
  uses: TBehavior
  instanceVariableNames: 'superclass methodDict format layout'
  classVariableNames: 'ClassProperties ObsoleteSubclasses'
  package: 'Kernel−Classes'
```

Example: Problem with class definition in Pharo

```
subclass: s uses: aT instanceVariableNames: names classVariableNames: cvNames
        category: cat
subclass: s uses: aT instanceVariableNames: names classVariableNames: cvNames
        package: cat
subclass: s instanceVariableNames: names classVariableNames: cvNames
        poolDictionaries: pools package: cat
subclass: s uses: aT instanceVariableNames: names classVariableNames: cvNames
        poolDictionaries: pools category: acat
...
...tag:...
...immediate:...
...ephemeron:...
```

Example of a fluid interface in Seaside

```
ScrapBook>>renderContentOn: html
  html paragraph: 'A plain text paragraph.'.
  html paragraph: [
    html render: 'A paragraph with plain text followed by a line break.'.
    html break.
    html emphasis: 'Emphasized text '.
    html render: 'followed by a horizontal rule.'.
    html horizontalRule.
    html render: 'An image: '.
    html image url: 'http://www.seaside.st/styles/logo-plain.png' ]
```

How to ensure that an instance is well initialized?

- Automatic initialize
- Lazy initialization
- Proposing the right interface
- Providing a default value

A course by Stéphane Ducasse
http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse
http://mooc.pharo.org