Instance Initialization

How to ensure that an instance is well initialized?

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

Provider Responsibility

This is the **responsibility** of the class to provide a **well-formed** object.

A client should not make **assumptions** or be responsible to send a **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
```

Packet class Definition

```
Packet class is automatically defined
Packet class
    instanceVariableNames: "

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 instance variable not initialized

-> error (for example, printOn:)
-> system fragile

Fragile Instance Creation Solutions

Automatic initialization of instance variables
Proposing a solid interface for the creation
Lazy initialization

Instance Initialization

How to ensure that an instance is well initialized?

**Automatic initialize**

Lazy initialization
Proposing the right interface
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

The New/Initialize Couple

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

(VW) Assuring Instance Variable

Problem: By default new class method returns
instance with uninitialized instance variables.

In VisualWorks, initialize method is not automatically
called by creation methods new/new:

How to initialize a newly created instance?

new calling initialize

Packet new
... should invoke the initialize method

Packet class>>new
  | inst |
  inst := super new.
  inst initialize.
  ^ inst
The New/Initialize Couple

Define an instance method that initializes the instance variables and override new to invoke it.

(1-2) Packet class>>new "Class Method"
     ^ super new initialize

(3) Packet>>initialize "Instance Method"
     super initialize.

(4) contents := 'default message'

Packet new (1-2) => aPacket initialize (3-4) => returning aPacket but initialized!
Reminder: You cannot access instance variables from a class

Instance Initialization

How to ensure that an instance is well initialized?

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

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
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]
  ^ contents

Better

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

is equivalent to

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

Instance Initialization

How to ensure that an instance is well initialized?

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

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

Providing a Default Value

OrderedCollection variableSubclass:
#SortedCollection

instanceVariableNames: 'sortBlock'
classVariableNames: 'DefaultSortBlock'

SortedCollection class>>initialize

DefaultSortBlock := [x :y | x <= y]

SortedCollection>>initialize
"Set the initial value of the receiver's sorting
algorithm to a default."

Instance Initialization

How to ensure that an instance is well initialized?

Automatic initialize
Lazy initialization
Proposing the right interface

Providing a default value
Providing a Default Value

SortedCollection class>>new: anInteger
   "Answer a new instance of SortedCollection. The default sorting is a <= comparison on elements."
   ^ (super new: anInteger) initialize

SortedCollection class>>sortBlock: aBlock
   "Answer a new instance of SortedCollection such
   ^ (super new: anInteger) initialize

Invoking per Default the Creation Interface

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

Forbidding new?

Problems: 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: anAddress
   ^ self new contents: aString ; addressee: anAddress
   raises an error

Packet class>>send: aString to: anAddress
   ^ super new contents: aString ; address: anAddress
Forbidding new

Solution: use basicNew and basicNew:

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

Conclusion: Never override basic* methods else you will not be able to invoke them later

---

How to Reuse Superclass Initialization?

A class>>new
  ^ super new doThat; andThat; end

B class>>forceClientInterface
  ^ self basicNew ???

**Solution:** Define the initialization behavior on the instance side

A>>doThatAndThatEnd
  ^ self doThat; andThat; end

A class>>new
  ^ super new doThatAndThatEnd

B class>>forceClientInterface
  ^ self basicNew doThatAndThatEnd

---

Even Better...Use initialize

But you cannot simply chain the calls...so use initialize

A>>initialize
  super initialize.
  self doThat; andThat; end

B>>initialize
  super initialize.
  self andFoo.

---

Different Self/Super

Do not invoke a super with a different method selector.

It's bad style because it links a class and a superclass.

It makes the code difficult to understand

This is dangerous in case the software evolves.
Example

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

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

*Use basicNew and basicNew:*

Super is static!

With the super foo:
- A new bar  -> 10
- B new bar  -> 10
- C new bar  -> 10

Without the super foo:
- A new bar  -> 10
- B new bar  -> 100
- C new bar