Some Points on Classes

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.iam.unibe.ch/~ducasse/

A template is proposed by the browser:
Smalltalk defineClass: #NameOfClass
   superclass: #(NameOfSuperclass)
   instanceVariableNames: 'instVarName1 instVarName2'
   classInstanceVariableNames: '
   imports: ''
   category: ''

Class Definition (VW)

Named Instance Variables

instanceVariableNames: 'instVarName1 instVarName2'

• Begins with a lowercase letter
• Explicity declared: a list of instance variables
• Name should be unique in the inheritance chain
• Default value of instance variable is nil
• Private to the instance: instance based (vs. C++ class-based)
• Can be accessed by all the methods of the class and its subclasses
• Instance variables cannot be accessed by class methods.
• A client cannot directly access instance variables.
• The clients must use accessors to access an instance variable.

Fill the Template (VW)

Smalltalk defineClass: #Packet
superclass: #(Object)
instanceVariableNames: 'contents addressee originator'

A template is proposed by the browser:
Smalltalk defineClass: #NameOfClass
   superclass: #(NameOfSuperclass)
   instanceVariableNames: 'instVarName1 instVarName2'
   classInstanceVariableNames: '
   imports: ''
   category: ''

Roadmap

• Class definition
• Method definition
• Basic class instantiation

Fill the Template (Sq)

Just fill this Template in:
! Object subclass: #Packet

classInstanceVariableNames: ''

Method Definition

• Fill in the template. For example:
  Packet>>defaultContents
  "returns the default contents of a Packet"
  ^ self addressee = aNode name

  Packet>>isAddressedTo: aNode
  "returns true if I'm addressed to the node aNode"
  ^ self addressee = aNode name
Object Instantiation

Objects can be created by:
- Direct Instance creation: `new/new`
- Messages to instances that create other objects
- Class specific instantiation messages

Some Naming Conventions

- Shared variables begin with an upper case letter
- Private variables begin with a lower case letter
- For accessors, use the same name as the instance variable accessed:
  
  `Packet>>addressee`  
  `^addressee`
  
  `Packet>>addressee: aSymbol`  
  `!addressee := aSymbol`

Methods always return a Value

- Message = effect + return value
- By default, a method returns itself
- In a method body, the `^` expression returns the value of the expression as the result of the method execution.

```smalltalk
Node>>accept: thePacket
  self send: thePacket
  ^self
```

Methods always return a value

- If we want to return the value returned by `#send`:

```smalltalk
Node>>accept: thePacket
  "self send: thePacket.
  self"
```

Some Naming Conventions

- Use imperative verbs for methods performing an action like `#openOn:`, `#close`, `#sleep`
- For predicates methods (returning a boolean) prefix the method with `is` or `has`
- Ex: `isNil`, `isAddressedTo:`, `isSentBy:`
- For converting methods prefix the method with `as`
- Ex: `asString`

Object Creation

- When a class creates an object = allocating memory + marking it to be instance of that class

Instance Creation with `new`

`aClass new`

returns a newly and `UNINITIALIZED` instance

```smalltalk
OrderedCollection new -> OrderedCollection ()
Packet new -> aPacket
```

Default instance variable values are nil
nil is an instance of UndefinedObject and only understands a limited set of messages

- Design Hint: Do not directly access instance variables of a superclass from subclass methods. This way classes are not strongly linked.

- Shared variables begin with an upper case letter
- Private variables begin with a lower case letter
- For accessors, use the same name as the instance variable accessed:
  
  `Packet>>addressee`  
  `^addressee`
  
  `Packet>>addressee: aSymbol`  
  `!addressee := aSymbol`

Accessing Instance Variables

Using direct access for the methods of the class

```smalltalk
Packet>>isSentBy: aNode
  "originator": aNode
```

is equivalent to use accessors

```smalltalk
Packet>>originator
  "^originator"
```

Packet>>isSentBy: aNode
  "self originator": aNode

- Message = effect + return value
- By default, a method returns itself
- In a method body, the `^` expression returns the value of the expression as the result of the method execution.

```smalltalk
Node>>accept: thePacket
  self send: thePacket
  ^self
```

Methods always return a value

- If we want to return the value returned by `#send`:

```smalltalk
Node>>accept: thePacket
  "self send: thePacket.
  self"
```

- Use `^` self to notify the reader that something abnormal is arriving

```smalltalk
MyClass>>foo
  "…
  ^self"
```

- Use `asString` for converting methods

```smalltalk
Packet>>addressee
  "^addressee"
```

```smalltalk
Packet>>addressee: aSymbol
  "!addressee := aSymbol"
```

Some Naming Conventions

- Use imperative verbs for methods performing an action like `#openOn:`, `#close`, `#sleep`
- For predicates methods (returning a boolean) prefix the method with `is` or `has`
- Ex: `isNil`, `isAddressedTo:`, `isSentBy:`
- For converting methods prefix the method with `as`
- Ex: `asString`
Opening the Box

1 to: 6 creates an interval
Number>>to: stop
   "Answer an Interval from the receiver up to the argument, 
   stop, with each next element computed by incrementing the 
   previous one by 1."
^Interval from: self to: stop by: 1

Strings...

1 printString
Object>>printString
   "Answer a String whose characters are a description   ... receiver."
| aStream |
aStream := WriteStream on: (String new: 16).
self printOn: aStream.
^ aStream contents

Instance Creation

1@2 creates a point
Number>>@ y
   "Answer a new Point whose x value is the receiver and   
   whose y value is the argument."
   <primitive: 18>
^ Point x: self y: y

Class-specific Messages

Array with: 1 with: 'lulu'
OrderedCollection with: 1 with: 2 with: 3
Rectangle fromUser -> 179@95 corner: 409@219
Browser browseAllImplementorsOf: #at:put:
Packet send: 'Hello mac' to: #mac
Workstation withName: #mac

new and new:
• new/basicNew: is used to specify the size of the 
  created instance
  Array new: 4 -> #(nil nil nil nil)
• new/new: can be specialized to define customized 
  creation
• basicNew/basicNew: should never be overridden
• #new/basicNew and new/basicNew: are class methods

Summary

How to define a class?
What are instance variables?
How to define a method?
Instances creation methods