Some Advanced Points on Classes

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

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."

   1 to: 6

Strings...

1 printString

   "Answer a String whose characters are a description of the receiver."

   | aStream |
   | aStream := WriteStream on: (String new: 16). |
   | self printOn: aStream. |
   ^ aStream contents

Object Instantiation

Objects can be created by:

- Direct Instance creation: new/new:
- Messages to instances that create other objects
- Class specific instantiation messages

Instantiation

• Basic class instantiation

Instance Creation with new

aClass new

returns a newly and UNINITIALIZED instance

   ^ Point x: self y: y

Messages to Instances

Messages to Instances that create Objects

1 to: 6 (an interval)
1@2 (a point)
(0@0) extent: (100@100) (a rectangle)
#lulu asString (a string)
1 printString (a string)
3 asFloat (a float)
#(23 2 3 4) asSortedCollection (a sortedCollection)

Instance Creation

1@2 creates a point

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

**Constraints**

- Classes defined using `#subclass:` support any kind of subclasses
- Classes defined using `#variableSubclass:` can only have: `variableSubclass:` or `variableByteSubclass:` subclasses
- Pointer classes and byte classes don't mix: e.g., only byte subclasses of byte classes.

**Variable size instance**

How do we represent objects whose size is variable such an array

- `Array new: 10`
- `Array new: 15`

**Two Views on Classes**

**Named** or **indexed** instance variables

- Named: `addresses` of Packets
- Indexed: `Array`

Or looking at them in another way:

- Objects with pointers to other objects
- Objects with variables of bytes (word, long)

Difference for efficiency reasons: arrays of bytes (like C strings) are faster than storing an array of pointers, each pointing to a single byte.

**Indexes on Classes**

- `ArrayedCollection variableSubclass: #Array`
- `instanceVariableNames: ''`
- `classVariableNames: ''`
- `poolDictionaries: ''`
- `category: 'Collections-Arrayed'`

Array new: 4 -> #((nil nil nil nil))

**Types of Classes**

- Indexed: `Named Definition Method Examples`
- No: `#subclass:`...
- Packet: yes
- String: yes
- `#isPointers, #isBits, #isBytes, #isFixed, #isVariable, #kindOfSubclass`

Method related to class types: `#isPointers, #isBits, #isBytes, #isFixed, #isVariable, #kindOfSubclass`

**Indexed Classes**

For classes that need a variable number of instance variables

- `ArrayedCollection variableSubclass: #Array`
- `instanceVariableNames: ''`
- `classVariableNames: ''`
- `poolDictionaries: ''`
- `category: 'Collections-Arrayed'`

Array new: 4 -> #((nil nil nil nil))

#(1 2 3 4) class isVariable -> true

**Indexed Classes**

Indexed variable is implicitly added to the list of instance variables

- Only one indexed instance variable per class
- Access with `#at:` and `#at:put:`
  - `#at:puck: answers the value, not the receiver`
  - Subclasses should also be indexed
### Roadmap
- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables

### The Meaning of is-a
A class defines the structure and the behavior of all its instances.

Each instance possesses its own set of values.

Instances share the behavior defined in their class with other instances via the instance of link.

### Index access
First access: anInstance at: 1
Size returns the number of indexed instance variables
Instantiated with #new:max

```smalltalk
| t |
t := (Array new: 4).
t at: 2 put: 'lulu'.
t at: 1 -> nil
```

### Classes and Objects
- Classes are objects too
- The same principle is true for objects and classes
- Same lookup strategy
- Everything that works at instance levels works at class level

- In some languages classes are not objects, still understanding it in Smalltalk will force you to really understand what instance/inheritance means

### Class Responsibilities
- Instance creation
- Class information (inheritance link, instance variables, method compilation,...)
- Examples:
  - Node allSubclasses -> OrderedCollection (Workstation OutputServer Workstation File)
  - LaserPrinter allInstances -> #() Node instVarNames -> #('name' 'nextNode') Workstation withName: #mac -> aWorkstation
  - Workstation selectors -> IdentitySet (#except: #originates:)
  - Workstation canUnderstand: #nextNode -> true

---

### Lookup...
1. Look in the classes
2. Go to the class

### The Meaning of is-a
- Every object is an instance of a class.
- When anObject receives a message, the method is looked up in its class
- And it continues possibly in its superclasses
- Every class is ultimately a subclass of Object (except Object)

Example: `macNode name`
macNode is an instance of Workstation
=> name is looked up in the class Workstation
name is not defined in Workstation
=> lookup continues in Node
name is defined in Node
=> lookup stops + method executed
A Class is an Object too…

- Every class (X) is the unique instance of its associated metaclass named X class
- Example:
  - Node is the unique instance of “Node class”
  - Point is the unique instance of “Point class”

Where is new defined?

- Node new: #node1
  - Node is an instance of Node class => new: is looked up in the class “Node class”
  - withName: is defined in the class “Node class” lookup stops + method executed

A Class is an Object too…

So messages sent to a class are looked up into the class of the class

Node withName: #node1
Node is an instance of “Node class”
- withName: is looked up in the class “Node class”
- withName: defined in “Node class” lookup stops + method executed

Lookup and Class Methods

- Workstation withName: #mac
  - Workstation is an instance of Workstation class
  - withName: is not defined in Workstation class lookup continues in the superclass of Workstation class = Node class
  - withName: is defined in Node class lookup stops + method executed

Class Parallel inheritance

- An object is a class if and only if it can create instances of itself.
- A Metaclass is just a class whose instances are classes
  - Point class is a metaclass as its instance is the class Point Variable

A Class is an Object too…

- Everything is an object
  - Each object is instance of one class
  - A class (X) is also an object, the sole instance of its associated metaclass named X class
  - An object is a class if and only if it can create instances of itself.
  - A Metaclass is just a class whose instances are classes
    - Point class is a metaclass as its instance is the class Point

Recap

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables

Roadmap

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables
**Advanced Classes**
- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- **Class Variables**

**Class Methods**
- As any object a (meta)class can have methods that represent the behavior of its instance: a class
- Uniformity => Same rules as for normal classes
- No constraint: just normal methods
- Can only access instance variable of the class:

**Class Instance Variables**
- Like any object, a class is an instance of a class that can have instance variables that represent the state of a class.
- When Point defines the new instance variable z, the instances of Point have 3 value (one for x, one for y, and one for z)
- When a metaclass defines a new instance variable, then its instance (a Class) gets a new value in addition to subclass, superclasses, methodDict...

**Class Method Examples**
- NetworkManager class>>new can only access uniqueInstance class instance variable and not instance variables (like nodes).
- Default Instance Creation class method:
  - new/new: and basicNew/basicNew:(see Direct Instance Creation)
  - Packets new
  - Specific instance creation method
  - Packet send: ‘Smalltalk is fun’ to: #lpr

**The Singleton Pattern**
- A class having only one instance
- We keep the instance created in an instance variable

```smalltalk
WebServer class
instanceVariableNames: 'uniqueInstance'

WebServer class>>new
self error: 'You should use uniqueInstance to get the unique instance'

WebServer class>>uniqueInstance
uniqueInstance ifNil:
    [ uniqueInstance := self basicNew initialize].
^ uniqueInstance
```

**Design Implications**
- An instance variable of a class can be used to represent information shared by all the instances of the class. However, you should use class instance variables to represent the state of the class (like the number of instances, ...) and not information of its instance.
- Should use sharedVariable instead (next Section).

**Singleton**
- WebServer being an instance of WebServer class has an instance variable named uniqueInstance.
- WebServer has a new value that is associated with uniqueInstance

**classVariable = Shared Variables**
- How to share state between all the instances of a class:
  - Use a classVariable
    - a classVariable is shared and directly accessible by all the instances of the class and subclasses
    - A pretty bad name: should have been called SharedVariables (now fixed in WW)
    - SharedVariable => begins with an uppercase letter
    - a classVariable can be directly accessed in instance methods and class methods

**classVariable = shared Variab. (Sq)**

**WebServer instanceVariableNames**
- 'uniqueInstance'

**Packet new:**
- Specific instance creation method
- `Packet send: 'Smalltalk is fun' to: #lpr`
Class Variables vs. Instance Variables

- a classVariable is shared and directly accessible by all the instances and subclasses
- Class instance variables, just like normal instance variables, can be accessed only via class message and accessors:
  - an instance variable of a class is private to this class.
- Take care: when you change the value of a classVariable
  the whole inheritance tree is impacted!

Example

- in the Scanner class a table describes the types of the characters (strings, comments, binary...). The original table is stored into a classVariable, its value is loaded into the instance variable. It is then possible to change the value of the instance variable to have a different scanner.

Object subclass: #Scanner
instanceVariableNames: 'source mark prevEnd hereChar token tokenType buffer typeTable'
classVariableNames: 'TypeTable'
category: 'System-Compiler-Public Access'

What you should know

- Classes are objects too
- Class methods are just methods on objects that are classes
- Classes are also represented by instance variables (class instance variables)
- (Shared Variables) ClassVariables are shared among subclasses and classes (metaclass)