Basic Objects, Conditionals and Loops

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

Conditionals and Loops

Booleans
Basic Loops
Overview of the Collection hierarchy—more than 80 classes: (Bag, Array, OrderedCollection, SortedCollection, Set, Dictionary...)
Loops and Iteration abstractions
Common object behavior

Booleans

• false and true are objects described by classes Boolean, True and False
• Uniform, but optimized and inlined (macro expansion at compile time)
• Logical Comparisons &, |, xor:, not
  • aBooleanExpr comparison anotherBooleanExpr
  • (1 isZero) & false
**Boolean Hierarchy**

- Please open your browser and analyse it
- How to implement in OO true and false without conditional?

```
Boolean
  not
  ifTrue:
```

- `True`
  - `not`
  - `ifTrue:`

- `False`
  - `not`
  - `ifTrue:`

**Boolean Lazy Logical Operators**

- Lazy Logical operators
  - `aBooleanExpr and: andBlock`
  - `andBlock` will only be valued if `aBooleanExpression` is true
  - `aBooleanExpression or: orBlock`
  - `orBlock` will only be valued if `aBooleanExpression` is false

```
false and: [1 error: 'crazy']
```

```
Prt-> false and not an error
```

**Conditional are Messages to Boolean**

- `aBoolean ifTrue: aTrueBlock ifFalse: aFalseBlock`
- `aBoolean ifFalse: aFalseBlock ifTrue: aTrueBlock`
- `aBoolean ifTrue: aTrueBlock`
- `aBoolean ifFalse: aFalseBlock`

- Hint: Take care — true is the boolean value and True is the class of true, its unique instance!

**Why Block Use in Conditional**

- Why do conditional expressions use blocks?
- Because, when a message is sent, the receiver and the arguments of the message are always evaluated. Blocks are necessary to avoid evaluating both branches.
Collections

A lot but key abstraction
Key iterators

Essential Collection

Sequenceable     ordered
ArrayedCollection fixed size + key = integer
Array            any kind of elements
CharacterArray   elements = character
String           
IntegerArray     arithmetique progression
Interval         dynamic chaining of the element
LinkedList       size dynamic + arrival order
OrderedCollection
SortedCollection  explicit order
Bag              possible duplicate + no order
Set              no duplicate + no order
IdentitySet      identification based on identity
Dictionary       element = associations + key based

Essential Collections: Another View

Collections

• Some criteria to identify them
  – Access: indexed, sequential or key-based.
  – Size: fixed or dynamic.
  – Element type: any or well-defined type.
  – Order: defined, defineable or none.
  – Duplicates: possible or not
Some Collection Methods

- Will be defined, redefined, optimized or forbidden in the subclasses
- Accessing: size, capacity, at: anInteger, at: anInteger put: anElement
- Testing: isEmpty, includes: anElement, contains: aBlock, occurrencesOf: anElement
- Adding: add: anElement, addAll: aCollection
- Removing: remove: anElement, remove: anElement ifAbsent: aBlock, remove: removeAll: aCollection
- Enumerating (See generic enumerating): do: aBlock, collect: aBlock, select: aBlock, reject: aBlock, detect: aBlock ifNone: aNoneBlock, inject: avalue into: aBinaryBlock
- Converting: asBag, asSet, asOrderedCollection, asSortedCollection, asArray, asSortedCollection: aBlock
- Creation: with: anElement, with: with:, with: with: with:, with: with: with: with: with:, with: All: aCollection

Sequenceable Specific (Array)

| arr |
arr := (calvin hates suzie).  
arr at: 2 put: loves.  
arr PrIt-> (calvin loves suzie)

- Accessing: first, last, atAllPut: anElement, atAll: anIndexCollection: put: anElement
- Searching (*: + ifAbsent:): indexOf: anElement, indexOf: anElement ifAbsent: aBlock
- Changing: replaceAll: anElement with: anotherElement
- Copying: copyFrom: first to: last, copyWith: anElement, copyWithout: anElement

KeyedCollection Specific (Dictionary)

| dict |
dict := Dictionary new.  
dict at: 'toto' put: 3.  
dict at: 'titi' ifAbsent: [4]. -> 4  
dict at: 'titi' put: 5.  
dict removeKey: 'toto'.  
dict keys -> Set ('titi')

- Accessing: at: aKey, at: aKey ifAbsent: aBlock, at: aKey ifAbsentPut: aBlock, at: aKey put: aValue, keys, values, associations
- Removing: removeKey: aKey, removeKey: aKey ifAbsent: aBlock
- Testing: includeKey: aKey

Common Shared Behavior
Common Shared Behavior

- Object is the root of the inheritance tree
- Defines the common and minimal behavior for all the objects in the system.
- Comparison of objects: ==, ~~, =, =~, isNil, notNil

Identity vs. Equality

- = anObject returns true if the structures are equivalent (the same hash number)
- (Array with: 1 with: 2) = (Array with:1 with:2) Print-> true
- == anObject returns true if the receiver and the argument point to the same object. == should never be overridden.

```
Object>>= anObject
  ^ self == anObject

~= is: not =
~~ is: not ==
```

- (Array with: 1 with: 2 ) == (Array with: 1 with:2) Print-> false
- (Array with: 1 with: 2 ) = (Array with: 1 with:2) Print-> true

Common Behavior: Printing

- Print and store objects: printString, printOn: aStream. printString calls printOn: aStream

```
(123 1 2 3) printString
-> ' (123 1 2 3)'
Date today printString
-> 'October 5, 1997'
```

Storing

- storeString, storeOn: aStream.
- storeString calls storeOn: aStream

```
Date today storeString
-> ' (Date readFromString: "10/5/1997")'
OrderedCollection new add: 4 ; add: 3 ; storeString
-> ' ( (OrderedCollection new) add: 4; add: 3; yourself)'
```

- You need the compiler, so for a deployment image this is not convenient
**readFromString: recreating Objects**

- Create instances from stored objects: class methods
  - `readFrom: aStream, readFromString: aString`

- `Object readFromString: '((OrderedCollection new)
  add: 4; yourself)'
- `-> OrderedCollection (4)`

**Notifying the Programmer**

- `error: aString,`
- `doesNotUnderstand: aMessage,`
- `halt, halt: aString,`
- `To invoke the debugger
  Input defaultState ifTrue:[self halt]`
- `shouldNotImplement`
- `Sign of bad design: subclassing`

**Copying in VW**

- Copying of objects: `shallowCopy, copy`
- `shallowCopy`: the copy shares instance variables with the receiver.
- `default implementation of copy is shallowCopy`

**Copying in VW**

- `Object>>copy`
  ^ `self shallowCopy postCopy`
- `Object>>postCopy`
  ^ `self`

- `postCopy is a hook method`
- `copy is a template method`
Summary

-timesRepeat:, to:do:, Array, OrderedCollection, Dictionary, Set
do:, select:, collect: