



Booleans, Conditionals and Loops

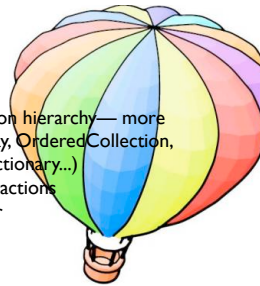
Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
<http://www.iam.unibe.ch/~ducasse/>

S.Ducasse

1

Outline

- 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



S.Ducasse

2



Boolean Messages

- Logical Comparisons: &, |, xor:, not
- **aBooleanExpr** comparison **aBooleanExpr**
 - (1 isZero) & false
 - Date today isRaining not
- Uniform, but optimized and inlined (macro expansion at compile time)

S.Ducasse

3



Boolean Lazy Logical Operators

- **Lazy** Logical operators
 aBooleanExpr **and:** andBlock
 andBlock will **only** be evaluated if aBooleanExpression is true
- aBooleanExpression **or:** orBlock
 orBlock will **only** be evaluated if aBooleanExpression is false
 false and: [1 error: 'crazy']
 Prlt -> false and not an error

S.Ducasse



Conditional: messages to booleans

- aBoolean **ifTrue:** aTrueBlock **ifFalse:** aFalseBlock
- aBoolean **ifFalse:** aFalseBlock **ifTrue:** aTrueBlock
- aBoolean **ifTrue:** aTrueBlock
- aBoolean **ifFalse:** aFalseBlock

```
(thePacket isAddressedTo: self)
  ifTrue: [self print: thePacket]
  ifFalse: [super accept: thePacket]
```

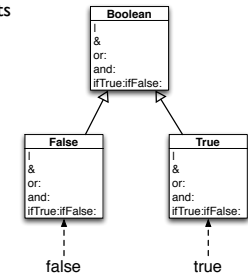
- Hint: Take care — true is the boolean value and True is the class of true, its unique instance!

S.Ducasse



Boolean Objects

- false and true are objects described by classes Boolean, True and False

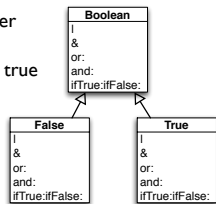


S.Ducasse



Boolean Hierarchy

- How to implement in OO true and false without conditional?
- Late binding: Let the receiver decide!
- Same message on false and true produces different results



S.Ducasse



Not

false not -> true
true not -> false

```
False>>not
"Negation -- answer true since the receiver is false."
^true
```

```
True>>not
"Negation--answer false since the receiver is true."
^false
```

S.Ducasse

8



| (Or)

- **true | true -> true**
- **true | false -> true**
- **true | anything -> true**
- **false | true -> true**
- **false | false -> false**
- **false | anything -> anything**

S.Ducasse

9



Boolean>> | aBoolean

Boolean>> | aBoolean

"Evaluating disjunction (OR). Evaluate the argument. Answer true if either the receiver or the argument is true."

self subclassResponsibility

S.Ducasse

10



False>> | aBoolean

false | **true** -> **true**

false | **false** -> **false**

false | **anything** -> **anything**

False>> | aBoolean

"Evaluating disjunction (OR) -- answer with the argument, aBoolean."

^ aBoolean

S.Ducasse

11



True>> | aBoolean

true | true -> **true**

true | false -> **true**

true | anything -> **true**

True>> | aBoolean

"Evaluating disjunction (OR) -- answer true since the receiver is true."

^ self

S.Ducasse

12



Boolean>>xor:

Boolean>>xor: aBoolean

"Exclusive OR. Answer true if the receiver is not equivalent to aBoolean."

^(self == aBoolean) **not**

Just implementing not on False and True defines xor: !

S.Ducasse

13



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.

S.Ducasse

14



To do

- Please open your browser and analyze it
- Have a look at the ifTrue:ifFalse: implementation

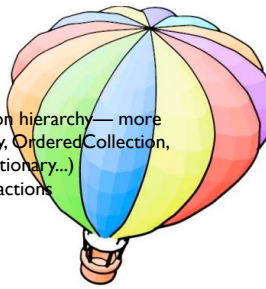
S.Ducasse

15



Roadmap

- 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



S.Ducasse

16



Some Basic Loops

- aBlockTest whileTrue
- aBlockTest whileFalse
- aBlockTest whileTrue: aBlockBody
- aBlockTest whileFalse: aBlockBody
- anInteger timesRepeat: aBlockBody
- [x<y] whileTrue: [x := x + 3]
- 10 timesRepeat: [Transcript show: 'hello'; cr]

S.Ducasse

17



For the Curious...

BlockClosure>>whileTrue: aBlock

^ self value

ifTrue: [aBlock value.
self whileTrue: aBlock]

BlockClosure>>whileTrue

^ [self value] whileTrue:[]

S.Ducasse

18



For the Curious...

```
Integer>>timesRepeat:aBlock  
"Evaluate the argument,aBlock,the number of  
times represented by the receiver."
```

```
| count |  
count := 1.  
[count <= self] whileTrue:  
    [aBlock value.  
    count := count + 1]
```

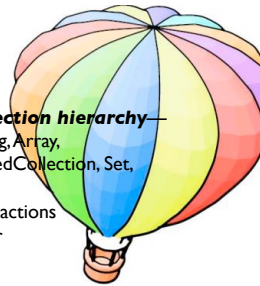
S.Ducas

19



Roadmap

- 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



S.Ducas

20



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, definable or none.
 - Duplicates: possible or not

S.Ducas

21



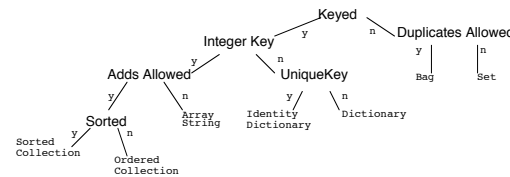
Essential Collection

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

S.Ducas



Essential Collections



S.Ducas

23



Some Collection Methods

- Are 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, removeAll: aCollection
- Enumerating (See generic enumerating): do: aBlock, collect: aBlock, select: aBlock, reject: aBlock, detect:, 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:, withAll: aCollection

S.Ducas



Array

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

S.Ducas



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')
```

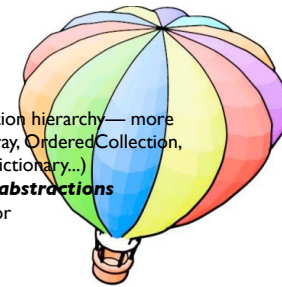
- Accessing: at: aKey, at: aKey ifAbsent: aBlock, at: aKey ifAbsentPut: aBlock, at: aKey put: aValue, keys, values, associations
- Removing: removeKey: aKey, removeKey: aKey ifAbsent: aBlock
- Testing: includeKey: aKey
- Enumerating: keysAndValuesDo: aBlock, associationsDo: aBlock, keysDo: aBlock

S.Ducas



Roadmap

- 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



S.Ducas

27



Choose your Camp!

To get all the absolute values of numbers you could write:

```
|result|
aCol := (2 -3 4 -35 4 -11).
result := aCol species new: aCol size.
1 to: aCollection size do:
  [ :each | result
    at: each put: (aCol at: each) abs].
result
```

S.Ducasse

28



Choose your Camp (II)

• You could also write:

```
(2 -3 4 -35 4 -11) collect: [ :each | each abs ]
```

• Really important: Contrary to the first solution, the second solution works well for indexable collections and also for sets.

S.Ducasse



Iteration Abstraction: do:/collect:

aCollection do: aOneParameterBlock
aCollection collect: aOneParameterBlock
aCollection with: anotherCollection do: aBinaryBlock

```
(15 10 19 68) do:
  [ :i | Transcript show: i printString ; cr ]
```

```
(15 10 19 68) collect: [ :i | i odd ]
Prt-> (true false true false)
```

```
(1 2 3) with: (10 20 30)
do: [ :x :y | Transcript show: (y ** x) printString ; cr ]
```

S.Ducasse



Opening the Box

Iterators are messages sent to collection objects
Collection is responsible of its traversal

SequenceableCollection>>do:aBlock
"Evaluate aBlock with each of the receiver's elements
as the argument."

```
1 to: self size do: [ :i | aBlock value: (self at: i) ]
```

S.Ducasse

31



select:/reject:/detect:

aCollection select: aPredicateBlock
aCollection reject: aPredicateBlock
aCollection detect: aOneParameterPredicateBlock
aCollection detect: aOneParameterPredicateBlock ifNone: aNoneBlock

```
(15 10 19 68) select: [ :i | i odd ] -> (15 19)
(15 10 19 68) reject: [ :i | i odd ] -> (10 68)
(12 10 19 68 21) detect: [ :i | i odd ] Prt-> 19
(12 10 12 68) detect: [ :i | i odd ] ifNone: [ 1 ] Prt-> 1
```

S.Ducasse



inject:into:

aCollection inject: aStartValue into: aBinaryBlock

```
| acc |
acc := 0.
(1 2 3 4 5) do: [ :element | acc := acc + element ].
acc
-> 15
Is equivalent to
(1 2 3 4 5)
  inject: 0
  into: [ :acc :element | acc + element ]
-> 15
```

Do not use it if the resulting code is not crystal clear!

S.Ducasse



Other Collection Methods

aCollection includes: anElement
aCollection size
aCollection isEmpty
aCollection contains: aBooleanBlock

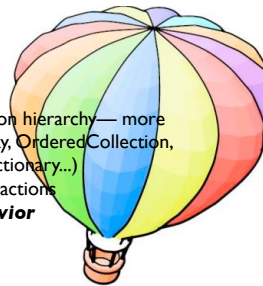
```
(1 2 3 4 5) includes: 4 -> true
(1 2 3 4 5) size -> 5
(1 2 3 4 5) isEmpty -> false
(1 2 3 4 5) contains: [ :each | each isOdd ] -> true
```

S.Ducasse



Roadmap

- 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**



S.Ducasse

35



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

S.Ducasse



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

- Take care when redefining `=`. One should override `hash` too!

S.Ducasse



Identity vs. Equality

- Remember pizza story?
- Identity
 - I want to eat your pizza
 - I eat it!
- Equality
 - I want to eat your pizza
 - I get a new that is like your pizza

S.Ducasse

38



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'
```

S.Ducasse



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

S.Ducasse



Recreating objects from strings

- Create instances from stored objects: class methods
`readFrom: aStream`, `readFromString: aString`
- `Object readFromString: '((OrderedCollection new) add: 4; yourself)'`
- `-> OrderedCollection (4)`

S.Ducasse



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
- `subclassResponsibility`
 - Abstract method

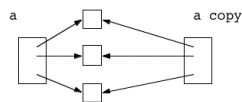
S.Ducasse

42



Copying

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



S.Ducasse

43



Copying

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

```
Object>>postCopy  
^ self
```

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

S.Ducasse



About responsibility

- No super magic global copy mechanism, just an object-oriented one
- The original object passes the control to its copy
- The copied object is in charge of its copy
- It decides which part should be copied and how

S.Ducasse

45



Summary

Booleans are objects too

Late binding

gives responsibility to the receiver to decide how to treat a message

the same message on different receiver produces different results

performs a dispatch, but the programmer does not do it, the execution does it!

Collections are objects too

Collections provides traversal abstractions

Objects share a common behavior