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

Conditional: messages to booleans
- aBoolean ifTrue: aTrueBlock ifFalse: aFalseBlock
- aBoolean ifTrue: aTrueBlock
- aBoolean ifFalse: aFalseBlock

(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

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)

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

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

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

(Or)
- true | true -> true
- true | false -> true
- true | anything -> true
- false | true -> true
- false | false -> false
- false | anything -> anything
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

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 ]

For the Curious...

BlockClosure>>whileTrue: aBlock
   ^ self value 
              ifTrue:
[ aBlock value. 
         !               self whileTrue: aBlock ]

BlockUse 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.

To do

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

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: !

False>> | aBoolean
  false | true -> true
  false | false -> false
  false | anything -> anything

True>> | aBoolean
  true | true -> true
  true | false -> true
  true | anything -> true

True>> | aBoolean
  "Evaluating disjunction (OR) -- answer true since the receiver is true."
  ^ self

False>> | aBoolean
  false | true -> true
  false | false -> false
  false | anything -> anything

False>> | aBoolean
  "Evaluating disjunction (OR). Evaluate the argument. Answer true if either the receiver or the argument is true."
  self subclassResponsibility

To do

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

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

```plaintext
(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
```

Opening the Box

Iterators are messages sent to collection objects. Collection is responsible of its traversal. SequenceableCollection>>do: aBlock

```plaintext
"Evaluate aBlock with each of the receiver's elements as the argument."
```

Choose your Camp!

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

```plaintext
[result]  
aCol := (2 3 4 35 4 -11).  
result := aCol map: [:each | each abs].  
result  
```

Choose your Camp (II)

- You could also write:
  ```plaintext
  (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.

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

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

Iteration Abstraction: do:/collect:

```plaintext
aCollection do: aOneParameterBlock
aCollection collect: aOneParameterBlock
```

```plaintext
aCollection with: anotherCollection do:
  aBinaryBlock
```

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

Inject:into:

```plaintext
aCollection inject: aStartValue into: aBinaryBlock
```

```plaintext
| acc |
| acc := 0. |
| (1 2 3 4 5) do: [:element | acc := acc * element]. |
| acc -> 15 |
```

Common object behavior

- Abstraction:
  ```plaintext
  Collection is responsible of its traversal.
  SequenceableCollection>>do: aBlock
  ```

```plaintext
Evaluate aBlock with each of the receiver's elements as the argument.
```

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

```plaintext
(in 10 19 68) select: [:i | i odd] ifNone: [1] result -> 19
```

```plaintext
(12 10 19 68) select: [:i | i odd] reject: [:i | i < 10] ifNone: [1] result -> 19
```

```plaintext
(15 19 68)reject: [:i | i odd] reject: [:i | i < 10] result -> 19
```

```plaintext
(12 10 19 68) detect: [:i | i odd] ifNone: [1] ifNone: [1] result -> 1
```

```plaintext
(15 10 19 68) select: [:i | i odd] collect: [:i | ifNone: [i] result -> 19
```

```plaintext
(12 10 19 68) detect: [:i | i odd] collect: [:i | ifNone: [i] result -> 1
```

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

- `anObject returns true if the structures are equivalent (the same hash value)`

  ```smalltalk
  (Array with: 1 with: 2) == (Array with: 1 with:2)
  \rightarrow true
  ```

- `anObject returns true if the receiver and the argument point to the same object. \textbf{should} never be overridden.`

  ```smalltalk
  self == anObject
  ```

Storing

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

  ```smalltalk
  Date today printString
  \rightarrow 'October 5, 1997'
  ```

  ```smalltalk
  (123 1 2 3) printString
  \rightarrow '(123 1 2 3)'
  ```

  ```smalltalk
  Date today printString
  \rightarrow 'October 5, 1997'
  ```

Recreating objects from strings

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

  ```smalltalk
  OrderedCollection new add: 4; add: 3; storeString
  \rightarrow '((OrderedCollection new) add: 4; add: 3; yourself)
  ```

  ```smalltalk
  OrderedCollection new add: 4; add: 3; yourselves`

Notifying the Programmer

- `error: aString,`
- `doesNotUnderstand: aMessage,`
- `halt, halt: aString,`

  ```smalltalk
  To invoke the debugger
  ```

  ```smalltalk
  Input defaultState ifTrue: [self halt]
  ```

  ```smalltalk
  shouldNotImplement
  ```

  ```smalltalk
  Sign of bad design: subclassing
  ```

  ```smalltalk
  subclassResponsibility
  ```

  ```smalltalk
  Abstract method
  ```

Copying

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

- `Object>>copy
  ^ self shallowCopy postCopy`

- `Object>>postCopy
  ^ self`

  ```smalltalk
  postCopy is a hook method
  ```

  ```smalltalk
  copy is a template method
  ```

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`
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