

Smalltalk Control Structures

Object

Boolean

True one instance - true

False one instance - false

Boolean operations

ifTrue: trueBlock ifFalse: falseBlock

ifTrue: trueBlock

ifFalse: falseBlock

& aBoolean

| aBoolean

Using Standard Control Structures

Find minimum of x and y

`min := x < y ifTrue: [x] ifFalse: [y]`

Boolean operations

Normal and - both arguments always evaluated

`(0 < i) & (i <= 100)`

Short-circuit and - second argument only evaluated when necessary

`(i <= array size) and: [(array at: i) > 0]`

Print i and i² for i between 1 and 10

```
i := 1
[i <= 10] whileTrue:
  [Transcript
   show: (i printString);
   show: '  ';
   show: (i squared printString); cr.
   i := i + 1]
```

Another way

```
1 to: 10 do: [:i |
  Transcript
  show: (i printString);
  show: '  ';
  show: (i squared printString);
  cr]
```

Evaluating Blocks

value	no arguments
value:	one argument
value:value:	two arguments

Implementing Standard Control Structures

True

ifTrue: trueBlock ifFalse: falseBlock

^trueBlock value

False

ifTrue: trueBlock ifFalse: falseBlock

^ falseBlock value

Implementing Standard Control Structures

Number

to: stop do: aBlock

| index |

index := start

[index <= stop]

whileTrue: [aBlock value: index.

index := index + 1]

Loops with Blocks

whileTrue: aBlock

^self value

ifTrue:

[aBlock value.

[self value]

whileTrue:

[aBlock value]]

Control Structures for Collections

Collections are data structures like Strings, Arrays, Sets, Lists, Dictionaries, etc.

Collection class has subclasses String, Array, Set, List, Dictionary, etc.

“Enumerating” protocol lets you operate on entire collection, taking a block as an argument.

Control Structures for Collections

numbers

do: [:each | each printOn: Transcript].

numbers

select: [:each | each > 100].

numbers

collect: [:each | each ^ 2]

(continued)

numbers

```
inject: 0 into: [:sum :i | sum + i]
```

names with: addresses

```
do: [:eachName :eachAddress |  
    letter name: eachName address:  
    eachAddress; print]
```

Abstract Class

class with no instances

template for subclasses, not instances.

defines standard interface

Important element of design reuse

Hard to invent good abstract classes

Abstract Classes and Their Function

Stream

Object and printing

Collection and iteration

Methods in an Abstract Class

Abstract method

method that **MUST** be defined by
subclasses

self subclassResponsibility

Template method

method defined in terms of abstract
methods

Stream - for writing

nextPut: anObject

nextPutAll: aCollection

aCollection

do: [:each | self nextPut: each]

cr

self nextPut: Character cr

Using WriteStreams for Printing

WriteStream of characters

aStream :=

WriteStream on: (String new: 20).

aStream nextPut: aCharacter.

aStream nextPutAll: aString.

aStream space; cr; tab.

aStream contents

Using WriteStreams for Printing

- Create WriteStream of characters
- Print on it
- Get the printed string
- Never use the original string that the WriteStream is streaming over, because it is abandoned when the stream grows.

Printing

- Object defines two methods that enable every object to print itself. Note that the contents of a WriteStream grows as needed.
- Public method - `printString`
- Method implemented by subclasses - `printOn:`

Printing in Object

printString

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

| aStream |

aStream :=

WriteStream on: (String new: 16).

self printOn: aStream.

^aStream contents

Printing in Object

printOn: aStream

"Append to the argument aStream a sequence of characters that identifies the receiver."

| title |

title := self class name.

aStream nextPutAll: ((title at: 1) isVowel
ifTrue: ['an '] ifFalse: ['a ']) , title

Printing

Classes customize printing by redefining `printOn:` and can inherit `printString` without change.

So, when you create a new class, define a `printOn:` method for it.

Concatenation

, (`comma`) is defined in `SequenceableCollection` and so is defined for `Strings`, `Arrays`, `OrderedCollections`, etc.

It concatenates the receiver with the argument.

`'this'` , `'is'` => `'this is'`

Concatenation

SequenceableCollection defines a rich set of copying operations, too many to list here. Never write a loop to copy elements to or from an array.

WARNING: Don't concatenate larger and larger strings, use WriteStreams.

Printing a Collection

printOn: aStream

"Append to the argument aStream a sequence of characters that identifies the collection. The general format for collections is "Collection-name (element element)" unless there are a large number in which case the listing is truncated with the words ...etc..."

(continued)

```
| tooMany |  
tooMany :=  
    aStream position + self maxPrint.  
aStream nextPutAll:  
    self class name, ' ('.
```

(continue)

```
self do:  
    [:element |  
        aStream position > tooMany  
            ifTrue: [aStream nextPutAll: '...etc...').  
                    ^self].  
        element printOn: aStream.  
        aStream space].  
aStream nextPut: $)
```

Printing a Paycheck

printOnCheckStream: aStream

aStream cr; cr.

aStream next: 40 put: (Character space).

DateFormat print: date on: aStream.

aStream cr.

(continued)

aStream nextPutAll: ((String new: 40 withAll:
Character space)

copyReplaceFrom: 1 to: employee name
size with: employee name).

AmountFormat print: amountPaid on:
aStream.

aStream cr; cr; cr; cr.

...

Printing Paychecks

paycheckString

| aStream |

aStream := WriteStream on: (String new:
20).

self printPaychecksOn: aStream.

^aStream contents

Transcript

Transcript is an instance of TextCollector.
TextCollector is not a subclass of Stream,
but supports a lot of WriteStream
protocol, such as nextPut:, nextPutAll:,
cr, space, tab

show: and print:

show: is like nextPutAll:

Transcript doesn't display any characters until it is sent show:. Look at show: method to see how to display otherwise.

print: anObject

anObject printOn: self

Uses of Inheritance

To be shared by all subclasses

- don't redefine

Default

- probably redefine

(continued)

Parameterized by subclasses (template method)

- rarely redefine
- change it by redefining the methods it calls

True abstract methods

- must redefine
- self subclassResponsibility

For an Abstract Class ...

Look for "subclassResponsibility" methods to see the core methods.

Look in subclasses to see examples.

Flow of control goes up and down the class hierarchy.

- (So don't try to follow it!)

Don't send "new" to the class.

Collection hierarchy

Collection ()

Bag ('contents')

SequenceableCollection ()

ArrayedCollection ()

LinkedList ('firstLink' 'lastLink')

Semaphore ('excessSignals')

Interval ('start' 'stop' 'end')

OrderedCollection ('firstIndex' 'lastIndex')

SortedCollection ('sortBlock')

Set ('tally')

Dictionary ()

IdentitySet ()

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

108

ArrayedCollection hierarchy

ArrayedCollection ()

Array ()

CharacterArray ()

String ()

ByteEncodedString ()

ByteString ()

Symbol ()

ByteSymbol ()

TwoByteSymbol ()

TwoByteString ()

Text ('string' 'runs')

RunArray ('runs' 'values' 'cacheRun' 'cacheRunStart')

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

109

Collection as an Abstract Class

Collection has no instance variables.

Collection defines as
subclassResponsibility

do:, add:, remove:, at:, at:put:

Collection as an Abstract Class

Template methods defined in terms of do:

select:, collect:, inject:into:,
detect:ifAbsent:, size

Abstract Classes

Abstract class as template

most operations defined in terms of do:

improved program skeleton

Abstract class as type

All collections understand same protocol
(do:, select:, collect, etc.)

Find an Element

detect: aBlock ifNone: exceptionBlock

"Evaluate aBlock with each of the receiver's elements as the argument. Answer the first element for which aBlock evaluates to true."

```
self do: [:each | (aBlock value: each)
                ifTrue: [^each]].
```

^exceptionBlock value

Accumulate a Value

inject: thisValue into: binaryBlock

"Accumulate a running value associated with evaluating the argument, binaryBlock, with the current value and the receiver as block arguments. The initial value is the value of the argument, thisValue."

Accumulate a Value

inject: thisValue into: binaryBlock |

nextValue |

nextValue := thisValue.

self do: [:each |

 nextValue := binaryBlock value:

nextValue value: each].

^nextValue

Transform a Collection

```
collect: aBlock  
  | newCollection |  
  newCollection := self species new.  
  self do: [:each | newCollection add:  
    (aBlock value: each)].  
  ^newCollection
```

Abstract Classes

Classes defined to be used **ONLY** as
superclasses.

Design of its subclasses.

Abstract methods are supposed to be
defined by subclasses.

Define standard interface that all
subclasses implement and any client
can use.