


# Traits: Tools and Methodology



Andrew P. Black

OGI School of Science & Engineering, OHSU  
Portland, Oregon, USA

Nathanael Schärli

Software Composition Group, IAM  
Universität Bern, Switzerland



# What are Traits?

- A programming language technology that enables reuse in place of duplication
  - Avoids problems of Multiple Inheritance & Mixins [ECOOP 2003 Analysis]
  - Allows programs to be smaller and more uniform [OOPSLA 2003 Refactoring]



# This talk:

- Is *not* primarily about traits
- It is about
  - the trait browser
  - the programming methodology developing around traits

# Traits and Uniform Protocol

- Protocol is a crucial idea in O-O
  - whether or not the language supports it
- Inheritance helps to *create* uniform protocol
  - a significant benefit to the user of a framework



# Smalltalk Enumeration Protocol

allSatisfy:	anySatisfy:	associationsDo:
collect:	collect:thenSelect:	count:
detect:	detect:ifNone:	detectMax:
detectMin:	detectSum:	difference:
do:	do:separatedBy:	do:without:
groupBy:having:	inject:into:	intersection:
noneSatisfy:	reject:	select:
select:thenCollect:	union:	

## Part of the interface of *Collection*

- implement internal iterators, *e.g.*,  
*aList select: [ :each | each isPrime ]*
- all subclasses of *Collection* share this protocol



# What about *Path*?

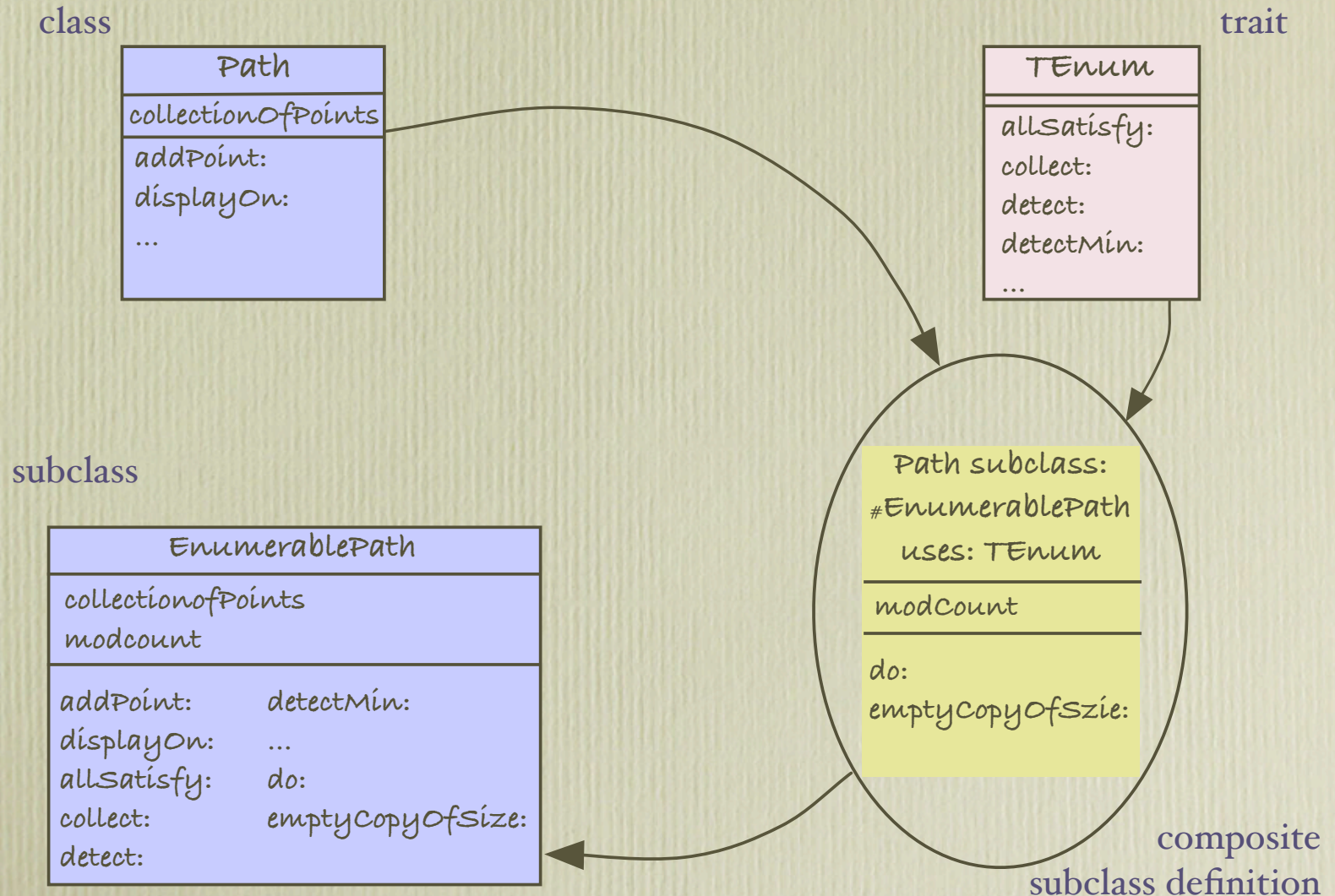
- A *Path* is a sequence of points
  - arcs, curves, lines, splines are all *Paths*
  - but *Path* is a subclass of *DisplayObject*, not of *Collection*
- *Path* does not implement the full enumeration protocol



# Traits in Smalltalk

- Smalltalk is a dynamically typed, class-based language with single inheritance
- Traits are “first class” collections of methods
  - Traits don't define state (instance variables)
  - Traits can be composed from sub-traits
  - A subclass can reuse methods from a trait as well as from a superclass

# Subclassing Path to create EnumerablePath





# What's the Payoff?

- 👁 We used traits to refactor the Smalltalk Collections classes [OOPSLA 2003]
  - 37 subclasses of *Collection* and 10 of *Stream*
- 👁 ... a total of 52 traits and 840 methods
  - one class used 22 traits!
- 👁 Refactored version had 10% fewer methods and 12% fewer bytes
  - In spite of 9% of methods being “too high” in original version



# Today's talk: two questions



# Today's talk: two questions

- How does the programmer manipulate traits ?



# Today's talk: two questions

- How does the programmer manipulate traits ?
  - Tools — the trait browser

# Today's talk: two questions

- How does the programmer manipulate traits ?
  - Tools — the trait browser
- How do traits change the way that programs are written?



# Today's talk: two questions

- How does the programmer manipulate traits ?
  - Tools — the trait browser
- How do traits change the way that programs are written?
  - Methodology



# The Trait Browser

👁 Two key ideas:

- ❑ Automatically and incrementally categorize methods in ways that help the programmer to see their inter-relationships
- ❑ Multiple views of a class: the extra level of structure provided by traits is optional



# Enumerations in the traits browser

The screenshot shows the 'Traits Browser: TCollEnumerationUI' window. The left pane lists various traits, with 'TCollEnumerationUI' selected. The middle pane shows the trait's methods, including 'select:'. The right pane shows the trait's documentation for 'select:'. Below the panes, there are buttons for '-own-' and '-super-'. The main area displays the documentation for 'select: aBlock', which includes a description and a code block.

**Traits Browser: TCollEnumerationUI**

TColl-Interfaces-Basic  
TColl-Interfaces-Common  
TColl-Interfaces-Common  
TColl-Classes  
TColl-Examples  
TColl-Tests  
TStream-Traits  
TStream-Classes  
TCollStream-Statistics

TCollEnumerationUI  
-own-  
TCollErrorsI  
TCollErrorsSizeIndependent  
TCollErrorsUI  
TCollExtensibleUI  
TCollMissfitsUI  
TCollPrintingUI

-- all --  
enumerating  
converting  
private  
copying  
-requires-

inject:into:  
intersection:  
noneSatisfy:  
reject:  
select:  
select:thenCollect:  
union:  
withIndexDo:

<- inst. ? class

-own- -super-

**select: aBlock**

"Evaluate aBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver, only those elements for which aBlock evaluates to true. Answer the new collection."

```
| newCollection |  
newCollection ← self emptyCopyOfSameSize.  
self withIndexDo: [:each :index |  
    (aBlock value: each) ifTrue: [newCollection unsafeAdd: each possiblyAt: index]].  
↑newCollection
```

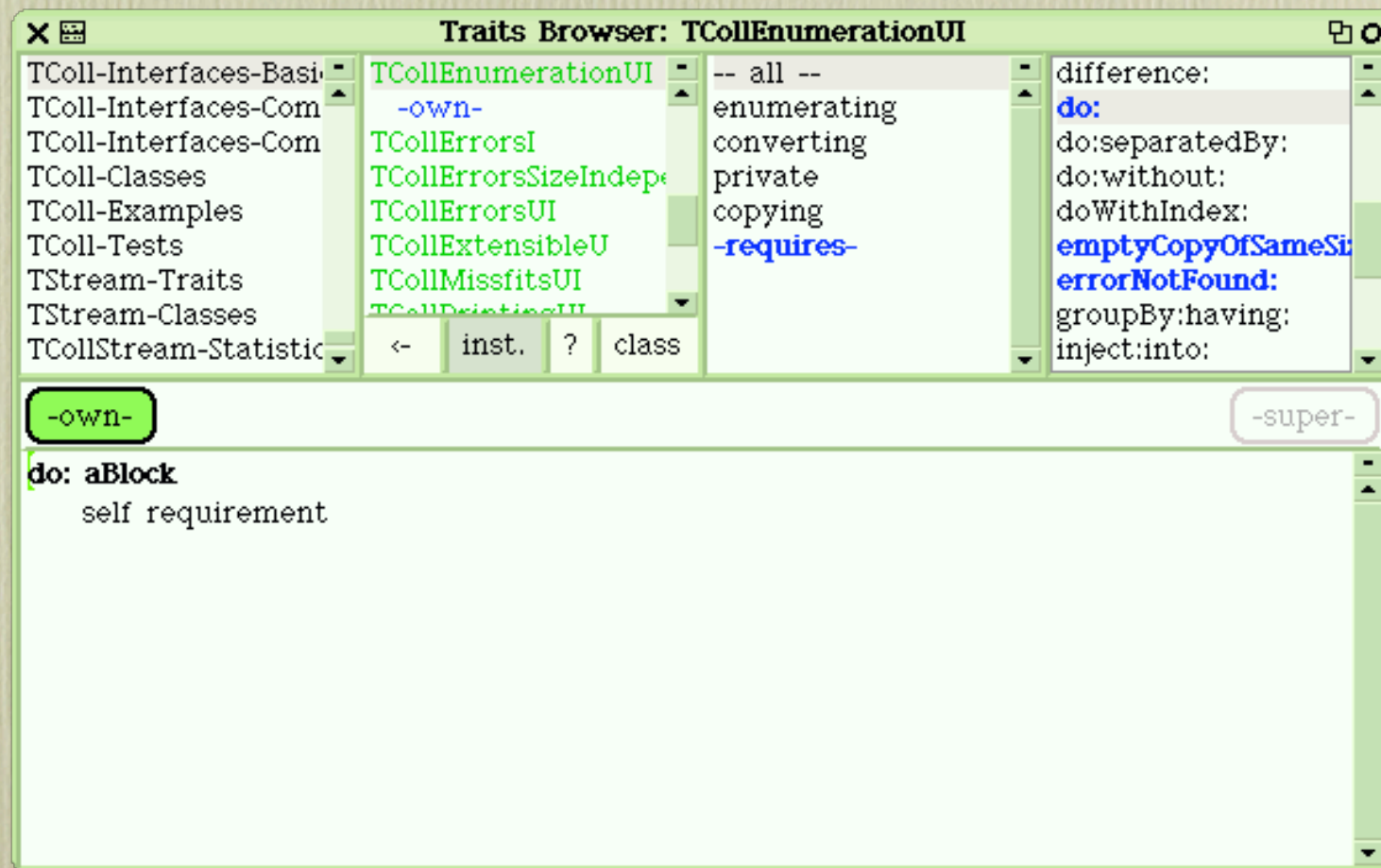
# Enumerations in the traits browser

The screenshot shows the 'Traits Browser: TCollEnumerationUI' window. The left sidebar lists various traits, with 'TCollEnumerationUI' selected. The main panel displays the trait's methods and properties, including 'inject:into:', 'intersection:', 'noneSatisfy:', 'reject:', 'select:', 'select:thenCollect:', 'union:', and 'withIndexDo:'. The 'withIndexDo:' method is highlighted. Below the main panel, there is a section for the 'withIndexDo: elementAndIndexBlock' method, which contains the following code:

```
withIndexDo: elementAndIndexBlock
| index |
index ← 1.
self do: [:each |
    elementAndIndexBlock value: each value: index.
    index ← index + 1].
```



# Enumerations in the traits browser



The screenshot shows the Traits Browser window titled "Traits Browser: TCollEnumerationUI". The left sidebar lists various traits and classes, including TColl-Interfaces-Basic, TColl-Interfaces-Common, TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, and TCollStream-Statistics. The main panel displays the hierarchy for TCollEnumerationUI, showing methods like enumerating, converting, private, copying, and requires. The right sidebar lists methods such as difference:, do:, do:separatedBy:, do:without:, doWithIndex:, emptyCopyOfSameSize, errorNotFound:, groupBy:having:, and inject:into:.

**Traits Browser: TCollEnumerationUI**

Left sidebar (selected): TColl-Interfaces-Basic, TColl-Interfaces-Common, TColl-Interfaces-Common, TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, TCollStream-Statistics.

Center panel (selected): TCollEnumerationUI, -own-, TCollErrorsI, TCollErrorsSizeIndependent, TCollErrorsUI, TCollExtensibleU, TCollMissfitsUI, TCollPrintingUI.

Right sidebar (selected): -- all --, enumerating, converting, private, copying, -requires-, difference:, do:, do:separatedBy:, do:without:, doWithIndex:, emptyCopyOfSameSize, errorNotFound:, groupBy:having:, inject:into:.

Buttons: <- inst. ? class

Buttons: -own- -super-

Content:

```
do: aBlock
  self requirement
```

# Enumerations in the traits browser

The screenshot shows the 'Traits Browser: TCollEnumerationUI' window. It features a tree view on the left with categories like 'TColl-Interfaces-Basic', 'TColl-Interfaces-Common', 'TColl-Classes', 'TColl-Examples', 'TColl-Tests', 'TStream-Traits', 'TStream-Classes', and 'TCollStream-Statistics'. The main area displays the traits of 'TCollEnumerationUI', including '-own-', 'TCollErrorsI', 'TCollErrorsSizeIndependent', 'TCollErrorsUI', 'TCollExtensibleUI', 'TCollMissfitsUI', and 'TCollPrintingUI'. A list of methods is shown on the right: '-- all --', 'enumerating', 'converting', 'private', 'copying', and '-requires-'. The bottom section shows the 'errorNotFound: index' method with a 'self requirement'.

**Traits Browser: TCollEnumerationUI**

TColl-Interfaces-Basic  
TColl-Interfaces-Common  
TColl-Interfaces-Common  
TColl-Classes  
TColl-Examples  
TColl-Tests  
TStream-Traits  
TStream-Classes  
TCollStream-Statistics

TCollEnumerationUI  
-own-  
TCollErrorsI  
TCollErrorsSizeIndependent  
TCollErrorsUI  
TCollExtensibleUI  
TCollMissfitsUI  
TCollPrintingUI

-- all --  
enumerating  
converting  
private  
copying  
-requires-

do:  
emptyCopyOfSameSize  
errorNotFound:

<- inst. ? class

-own- -super-

**errorNotFound: index**  
self requirement



# Enumerations in the traits browser

The screenshot shows the 'Traits Browser: TCollEnumerationI' window. The left sidebar lists various traits and classes, with 'TCollEnumerationUI' selected. The main pane displays the 'do:' method for 'aBlock'.

**Traits Browser: TCollEnumerationI**

**Left Sidebar:**

- TColl-Interfaces-Basic
- TColl-Interfaces-Collection
- TColl-Interfaces-Collection
- TColl-Classes
- TColl-Examples
- TColl-Tests
- TStream-Traits
- TStream-Classes
- TCollStream-Statistics

**Selected Trait: TCollEnumerationUI**

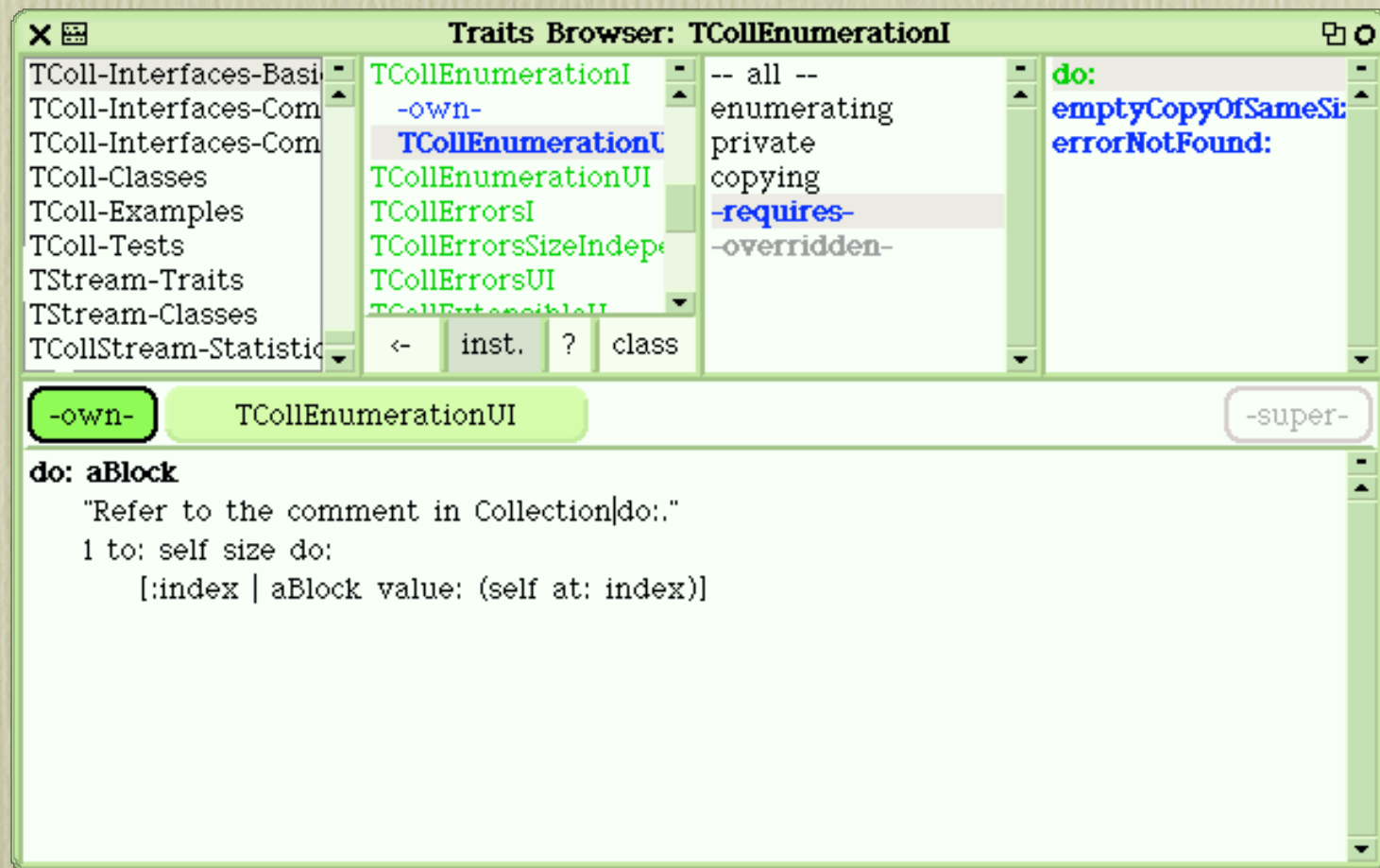
**Methods:**

- all --
- enumerating
- private
- accessing
- error handling
- copying
- requires-
- supplies-
- overrides-
- detectMax:
- detectMin:
- detectSum:
- difference:
- do:
- do:separatedBy:
- do:without:
- doWithIndex:
- findBinary:

**Method Definition:**

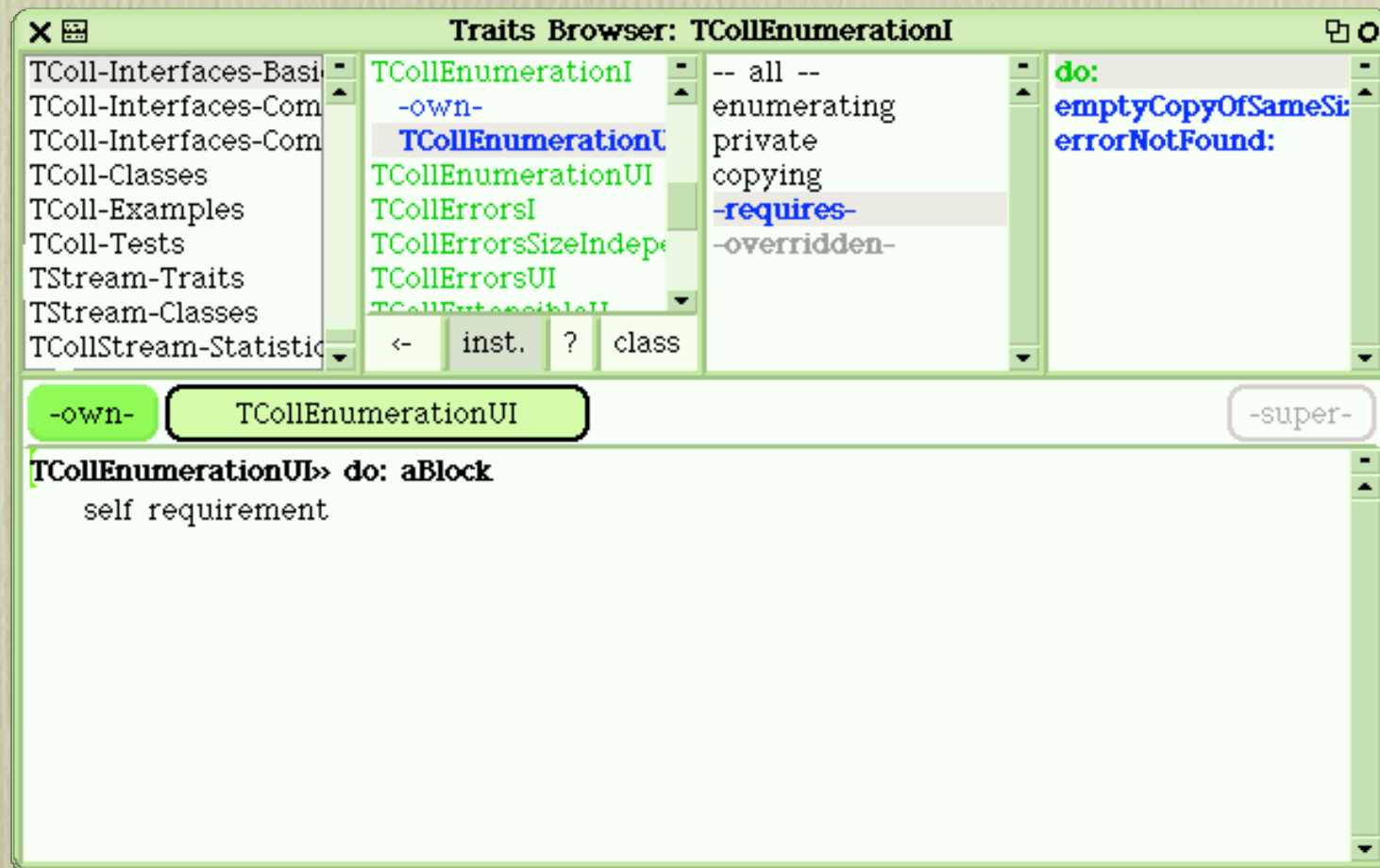
```
do: aBlock
  "Refer to the comment in Collection|do:."
  1 to: self size do:
    [:index | aBlock value: (self at: index)]
```

# Enumerations in the traits browser





# Enumerations in the traits browser



# overrides virtual category

The screenshot shows a window titled "Traits Browser: CollHeap". It contains a tree view on the left with categories like "TColl-Interfaces-Basic", "TColl-Interfaces-Common", "TColl-Classes", "TColl-Examples", "TColl-Tests", "TStream-Traits", "TStream-Classes", and "TCollStream-Statistics". The "CollHeap" class is selected in the tree. Below the tree, there are buttons for "-own-", "TCollHeapImpl", and "-super-". The main area displays the "capacity" method with the description "Answer the current capacity of the receiver." and a note "↑ self array size".

Category	Class	Method	Method
TColl-Interfaces-Basic	CollArray	adding	at:
TColl-Interfaces-Common	CollBag	copying	atRandom:
TColl-Interfaces-Common	CollDictionary	comparing	basicHasEqualElement
TColl-Classes	CollExtensibleSequencer	enumerating	capacity
TColl-Examples	CollExtensibleSequencer	private-heap	collect:
TColl-Tests	CollExtensibleUnsequencer	<del>supplies</del>	copyWith:
TStream-Traits	CollHeap	<del>overrides</del>	do:
TStream-Classes		<del>sending super-</del>	implementation
TCollStream-Statistics			remove;ifAbsent:

**capacity**  
"Answer the current capacity of the receiver."  
↑ self array size



# overrides virtual category

The screenshot shows a window titled "Traits Browser: CollHeap". It contains a tree view of traits on the left, a list of traits in the center, and a list of methods on the right. The "CollHeap" trait is selected in the center list. Below the lists, there are buttons for "-own-", "TCollHeapImpl", and "-super-". The main area displays the details for the "capacity" trait of "TCollMissfitsUI", including its description and a reference to "self size".

Left Panel (Tree View)	Center Panel (Trait List)	Right Panel (Method List)
TColl-Interfaces-Basi	CollArray	adding
TColl-Interfaces-Cor	CollBag	copying
TColl-Interfaces-Cor	CollDictionary	comparing
TColl-Classes	CollExtensibleSequer	enumerating
TColl-Examples	CollExtensibleSequer	private-heap
TColl-Tests	CollExtensibleUnsequ	-supplies-
TStream-Traits	CollHeap	-overrides-
TStream-Classes		-sending super-
TCollStream-Statistic		

Buttons: -own- TCollHeapImpl -super-

**TCollMissfitsUI» capacity**  
"Answer the current capacity of the receiver."  
↑ self size

# sending-super virtual category

- Contains all the methods in this class or trait that make *super*-sends



# sending-super virtual category

The screenshot shows a window titled "Traits Browser: CollHeap". It has a tree view on the left with categories like TColl-Interfaces-Basic, TColl-Interfaces-Common, TColl-Interfaces-Common, TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, and TCollStream-Statistics. The middle pane shows a list of traits: CollArray, CollBag, CollDictionary, CollExtensibleSequencer, CollExtensibleSequencer, CollExtensibleUnsequencer, and CollHeap. The right pane shows a list of traits: adding, copying, comparing, enumerating, private-heap, -supplies-, -overrides-, and -sending super-. The bottom pane shows the implementation of the -sending super- trait for TCollHeapImpl, which is a subclass of TCollSortBlockBasedImpl. The implementation is as follows:

```
TCollSortBlockBasedImpl» = aCollection  
"Answer true if my and aCollection's species are the same,  
and if our blocks are the same, and if our elements are the same."  
  
↑ self implementation = aCollection implementation and:  
  [self sortBlock = aCollection sortBlock and: [super = aCollection]]
```

# sending-super virtual category

The screenshot shows the 'Traits Browser: CollHeap' window. The left pane lists various traits and classes, with 'CollHeap' selected. The middle pane shows the 'CollHeap' class with buttons for '<-', 'inst.', '?', and 'class'. The right pane shows a list of virtual categories: 'adding', 'copying', 'comparing', 'enumerating', 'private-heap', '-supplies-', '-overrides-', and '-sending super-'. The '-sending super-' category is selected. Below the panes, there are buttons for '-own-' and '-super-'. The main area displays the definition of the 'TCollBasicImpb' virtual category, which is a subclass of 'aCollection'.

**Traits Browser: CollHeap**

TColl-Interfaces-Basi  
TColl-Interfaces-Com  
TColl-Interfaces-Com  
TColl-Classes  
TColl-Examples  
TColl-Tests  
TStream-Traits  
TStream-Classes  
TCollStream-Statistic

CollArray  
CollBag  
CollDictionary  
CollExtensibleSequer  
CollExtensibleSequer  
CollExtensibleUnsequ  
CollHeap

<- inst. ? class

adding  
copying  
comparing  
enumerating  
private-heap  
-supplies-  
-overrides-  
-sending super-

-own- TCollHeapImpl -super-

**TCollBasicImpb» = aCollection**

"Answer true if my species and aCollection species are equal, and if our starts, steps and sizes are equal."

self == aCollection ifTrue: [↑ true].  
aCollection species = self species ifFalse: [↑ false].  
↑ self hasEqualElements: aCollection.



# Trait conflicts

- Sibling traits with different methods on the same message generate a conflict
  - The programmer must resolve it explicitly

# Trait conflicts

The screenshot shows a window titled "Traits Browser: DemonstrateConflict". It features a sidebar on the left with a list of trait categories: TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, TCollStream-Statistic, Andrew-Interface, Traits-Ex-ReadWrite, Browser-HiddenMet, and Traits-ConflictExamp. The main area is divided into several panes. The top-left pane shows a list of traits: DemonstrateConflict, -own-, TCircle, and TColor. The top-right pane shows a list of trait methods: -- all --, comparing, geometry, accessing, -requires-, -conflicts-, -supplies-, -overridden-, and -overrides-. The bottom-left pane shows a list of trait methods: hash. The bottom-right pane shows a list of trait methods: hash. The bottom of the window has a navigation bar with buttons: -own-, TCircle, TColor, and -super-. Below the navigation bar, the "hash" trait is selected, and its definition is shown: self traitConflict.

Traits Browser: DemonstrateConflict

TColl-Classes  
TColl-Examples  
TColl-Tests  
TStream-Traits  
TStream-Classes  
TCollStream-Statistic  
Andrew-Interface  
Traits-Ex-ReadWrite  
Browser-HiddenMet  
Traits-ConflictExamp

DemonstrateConflict  
-own-  
TCircle  
TColor

-- all --  
comparing  
geometry  
accessing  
-requires-  
-conflicts-  
-supplies-  
-overridden-  
-overrides-

hash

<- inst. ? class

-own- TCircle TColor -super-

hash  
self traitConflict



# Programming Methodology

• Class hierarchy takes on many roles in ordinary O-O programming:

1. conceptual classification
2. definition of protocols (interfaces)
3. modularization
4. reuse of implementations
5. incremental modification



# Conceptual classification suffers

- 🌀 It's difficult or impossible to reconcile all of these roles
  - 🌀 Corrupting the conceptual relationship does not create immediate problems!
    - ❑ The problems are longer term, as the program ceases to model the domain
- ➡ Reuse takes priority over modeling



# Traits avoid this problem

- Traits support modularization directly (3)
- Trait methods can be reused anywhere in a hierarchy (4)
- Inheritance with traits allows reuse of the  $\square$  (5)
- Traits make protocol concrete, and make it easy to implement uniform interfaces (2)

# Traits avoid this problem



# Traits avoid this problem

- ➡ The class hierarchy is now free to be used for conceptual classification



# Uniform Protocol

- In conventional O-O programming, inheritance is the *only* tool available for making protocol uniform
  - If inheritance is used for another purpose, uniformity suffers
  - Programmer must build-up protocol one method at a time
- Traits allow classes to be constructed by *protocol composition*



# Uncovering Hidden Structure

- Many classes implement multiple protocols
- These protocols are rarely distinguished
  - Java's implements and interface keywords are under-used
  - Smalltalk's protocol categorization is only for documentation
- Trait browser lets us reify protocol after the fact



# Traits and Agile Methodologies

## 👁 XP and trait programming share practices

- ❑ continuous design
- ❑ refactoring
- ❑ testing
- ❑ pair programming
- ❑ collective ownership



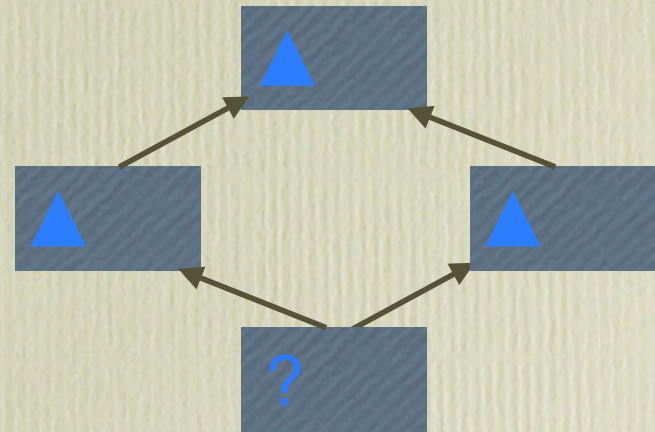
# Tools and methodology interact

- Methodology without tool support □ pious hope
- Tools without methodology □ too much rope
- Trait language features and browser co-evolved with the methodology



# Explicit conflict resolution

- Multiple inheritance characterized by complex rules for “automatic” conflict resolution.
  - superclass precedence
  - diamond problem with multiply inherited state
- Trait conflicts must be resolved explicitly
  - Browser makes it easy





# Fixing a conflict

The screenshot shows the 'Traits Browser: DemonstrateConflict' window. The left sidebar lists various trait collections, including 'TColl-Interfaces-Com', 'TColl-Classes', 'TColl-Examples', 'TColl-Tests', 'TStream-Traits', 'TStream-Classes', 'TCollStream-Statistic', and 'Andrew-Interface'. The main panel displays the 'DemonstrateConflict' trait, which includes a list of selectors: '-own-', 'TColor', 'TCircle', and 'EnumRectangle'. The 'hash' selector is highlighted in red. Below the main panel, there are buttons for '-own-', 'TColor', and '-super-'. A tooltip is visible over the 'TColor' button, containing the text 'Set exclusion' and 'Remove selector from trait: TColor'. The bottom panel shows the 'hash' selector with the code 'self traitConflict'.

**Traits Browser: DemonstrateConflict**

TColl-Interfaces-Com  
TColl-Classes  
TColl-Examples  
TColl-Tests  
TStream-Traits  
TStream-Classes  
TCollStream-Statistic  
Andrew-Interface

DemonstrateConflict  
-- all --  
comparing  
color  
-requires-  
-conflicts-  
-overridden-  
-overrides-

hash  
=

<- inst. ? class

-own- TColor -super-

Set exclusion  
Remove selector from trait: TColor

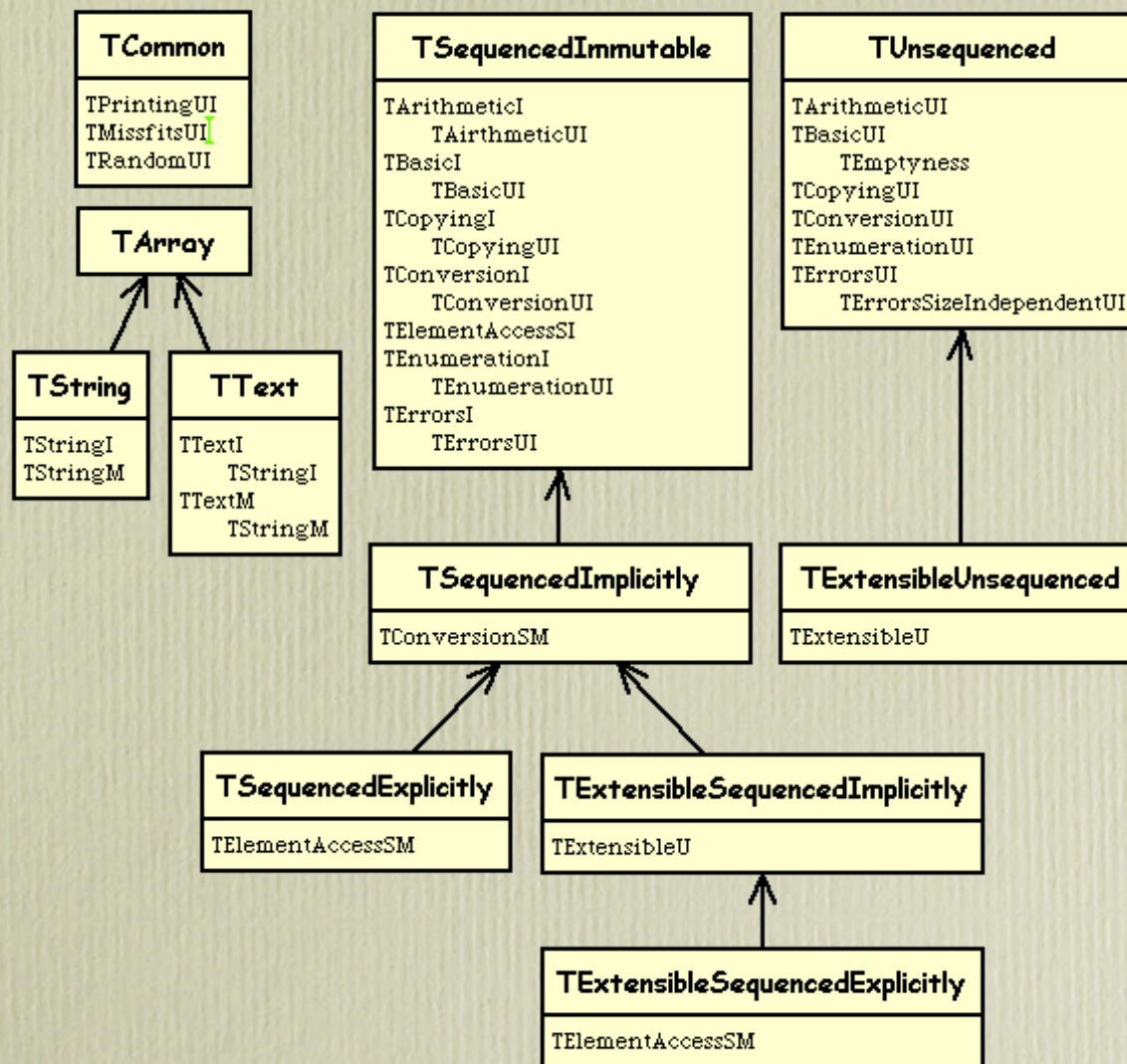
hash  
self traitConflict

# Flattening

- A class composed from traits can be viewed as if it were “flat”
  - the traits are “inlined”
- Extra structure provided by traits is always optional
  - `super` is not bound until a trait is used.
  - no “rename” operation
- A class can be built from a *score* of traits



# Trait nesting in Collections



# Conclusion (I/2)

- Combination of (Traits Language + Traits Browser) is a valuable tool
  - multiple views on a program
  - delayed decision making
  - late extraction of traits



# Conclusion (2/2)

- Raised the level of abstraction of the programming process
  - Programming with whole protocols rather than single methods
  - Visible requirements & overrides, and explicit conflict resolution, help avoid bugs