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

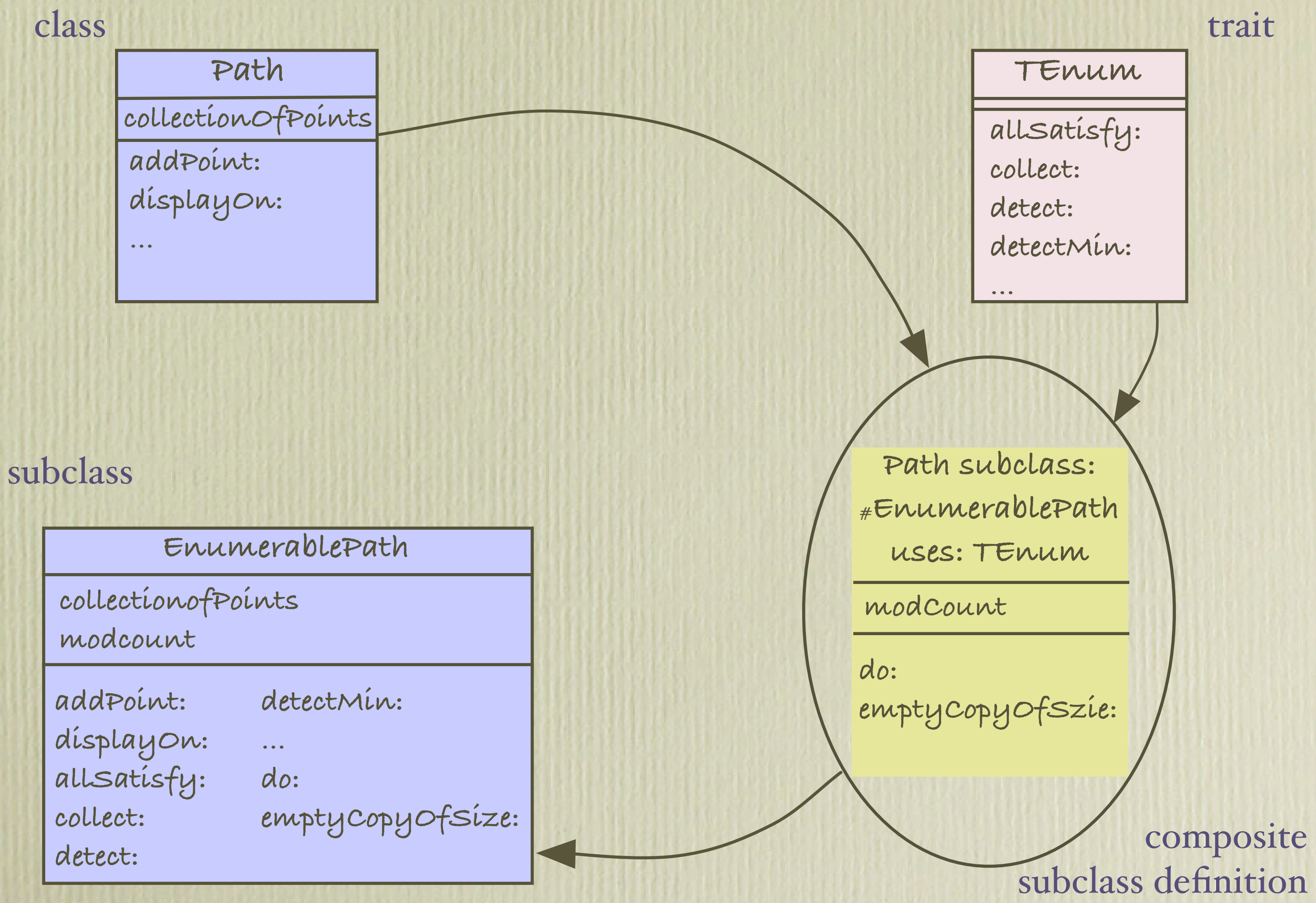
What about Path?

- 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?

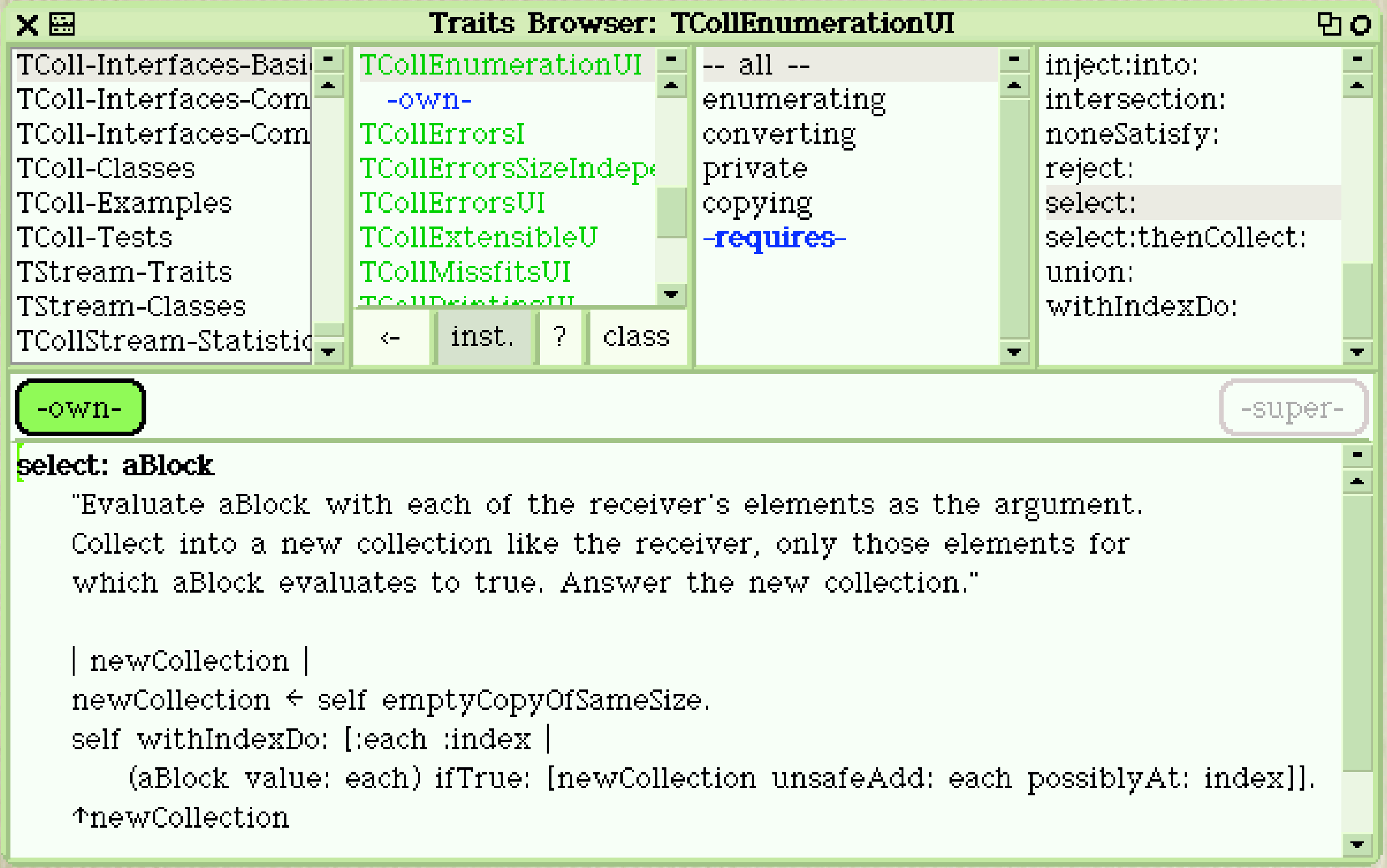
Today's talk: two questions

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

The Trait Browser

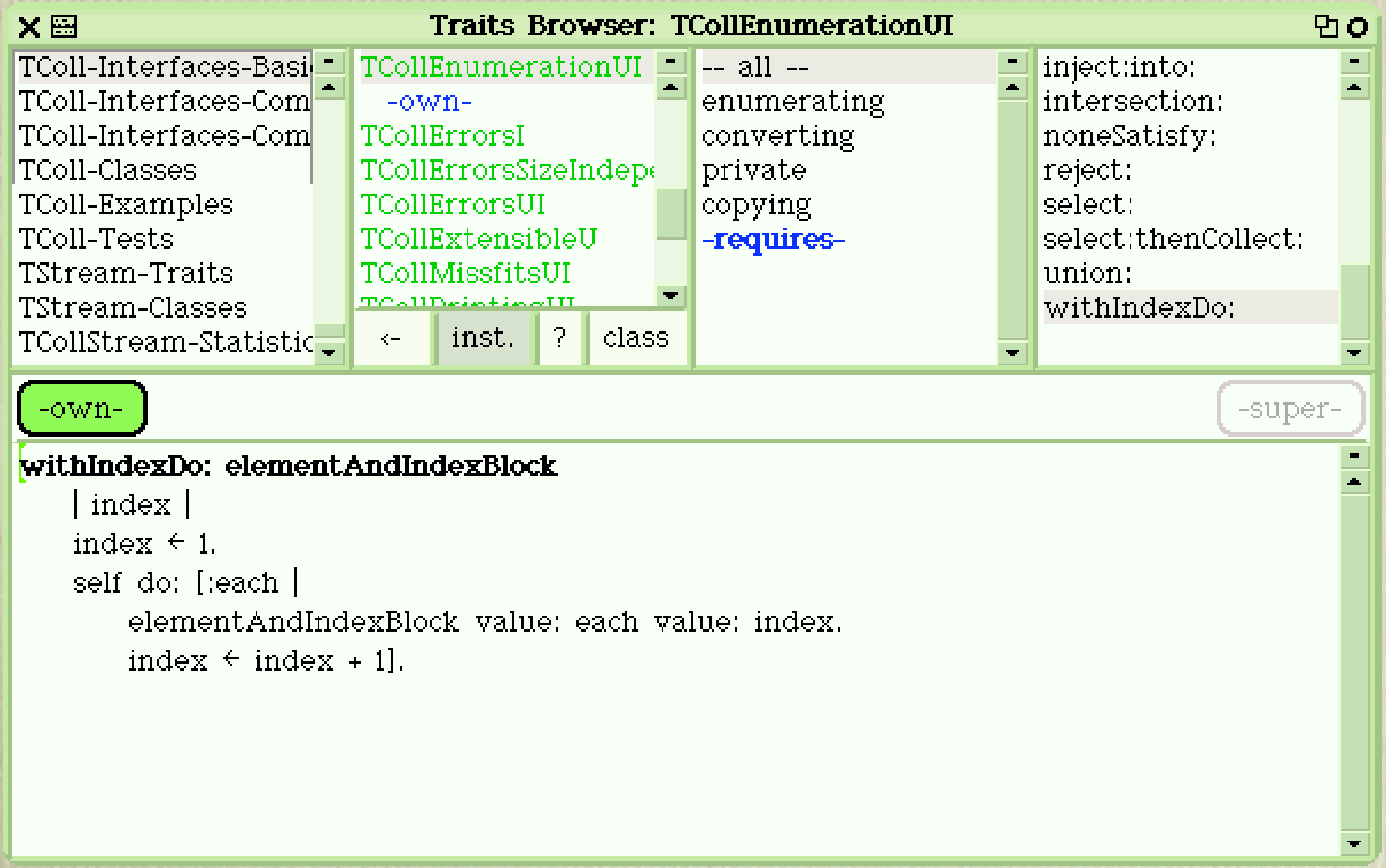
- Two key ideas:
  - Automatically and incrementally categorize methods in ways that help the programmer to see their inter-relationships
  - Multiple views of a class: the extra level of structure provided by traits is optional

Enumerations in the traits browser



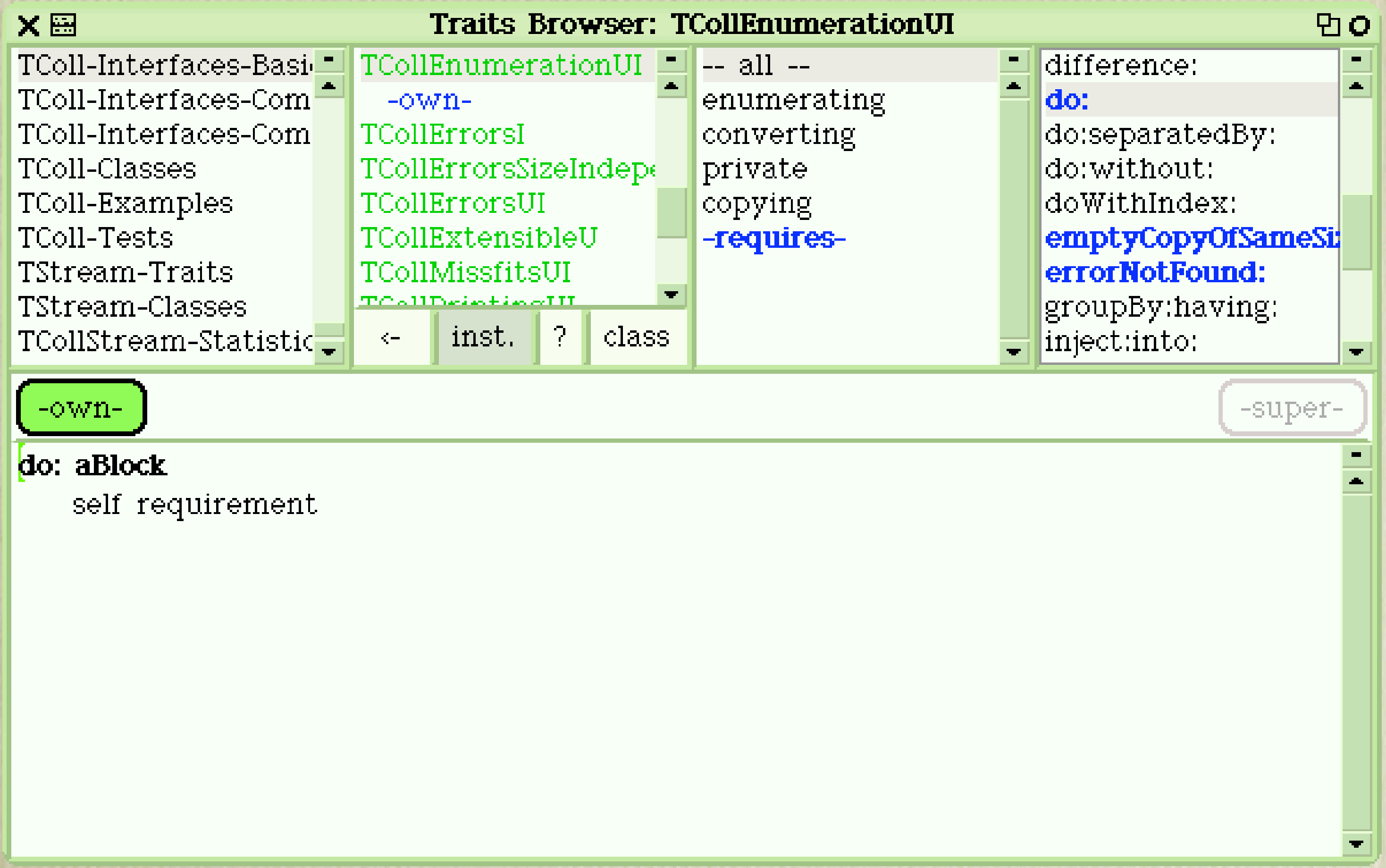
```
select: aBlock
    "Evaluate aBlock with each of the receiver's elements as the argument.
    Collect into a new collection like the receiver, only those elements for
    which aBlock evaluates to true. Answer the new collection."

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

Enumerations in the traits browser



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

Enumerations in the traits browser



Traits Browser: TCollEnumerationUI

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

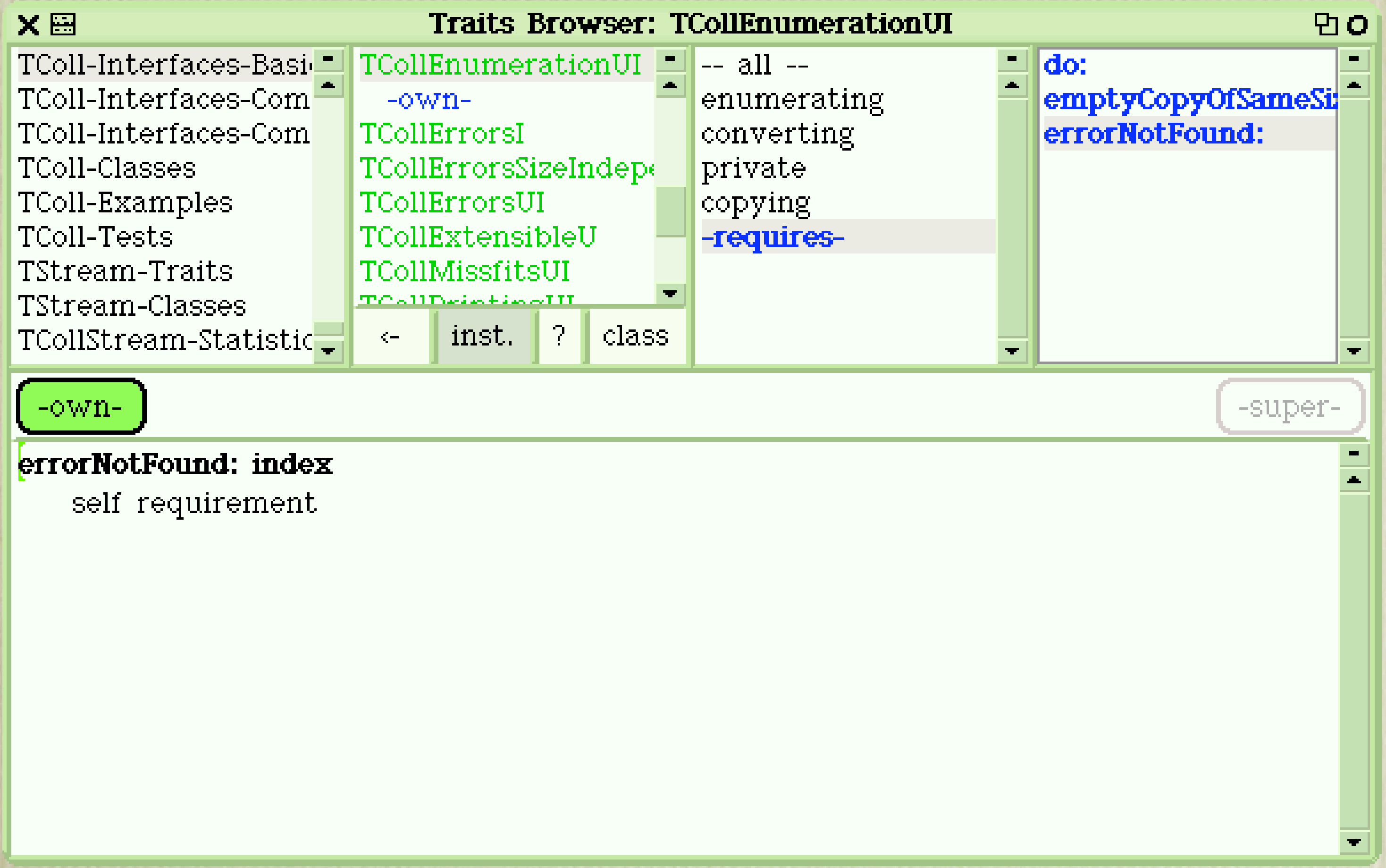
TCollEnumerationUI
-own-
TCollErrorsI
TCollErrorsSizeIndepe
TCollErrorsUI
TCollExtensibleU
TCollMissfitsUI

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

difference:
do:
do:separatedBy:
do:without:
doWithIndex:
emptyCopyOfSameSi
errorNotFound:
groupBy:having:
inject:into:

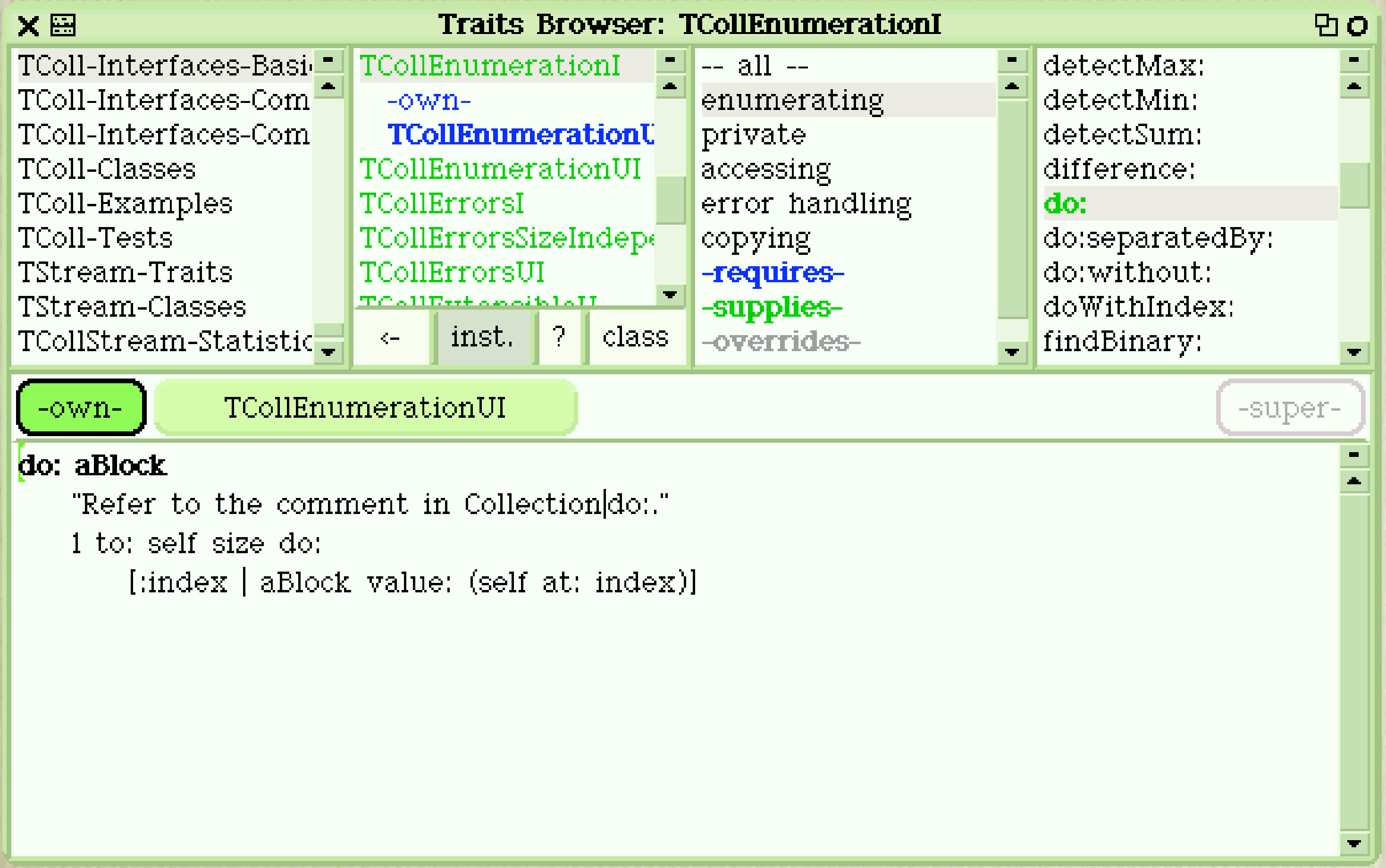
```
do: aBlock
	self requirement
```

Enumerations in the traits browser



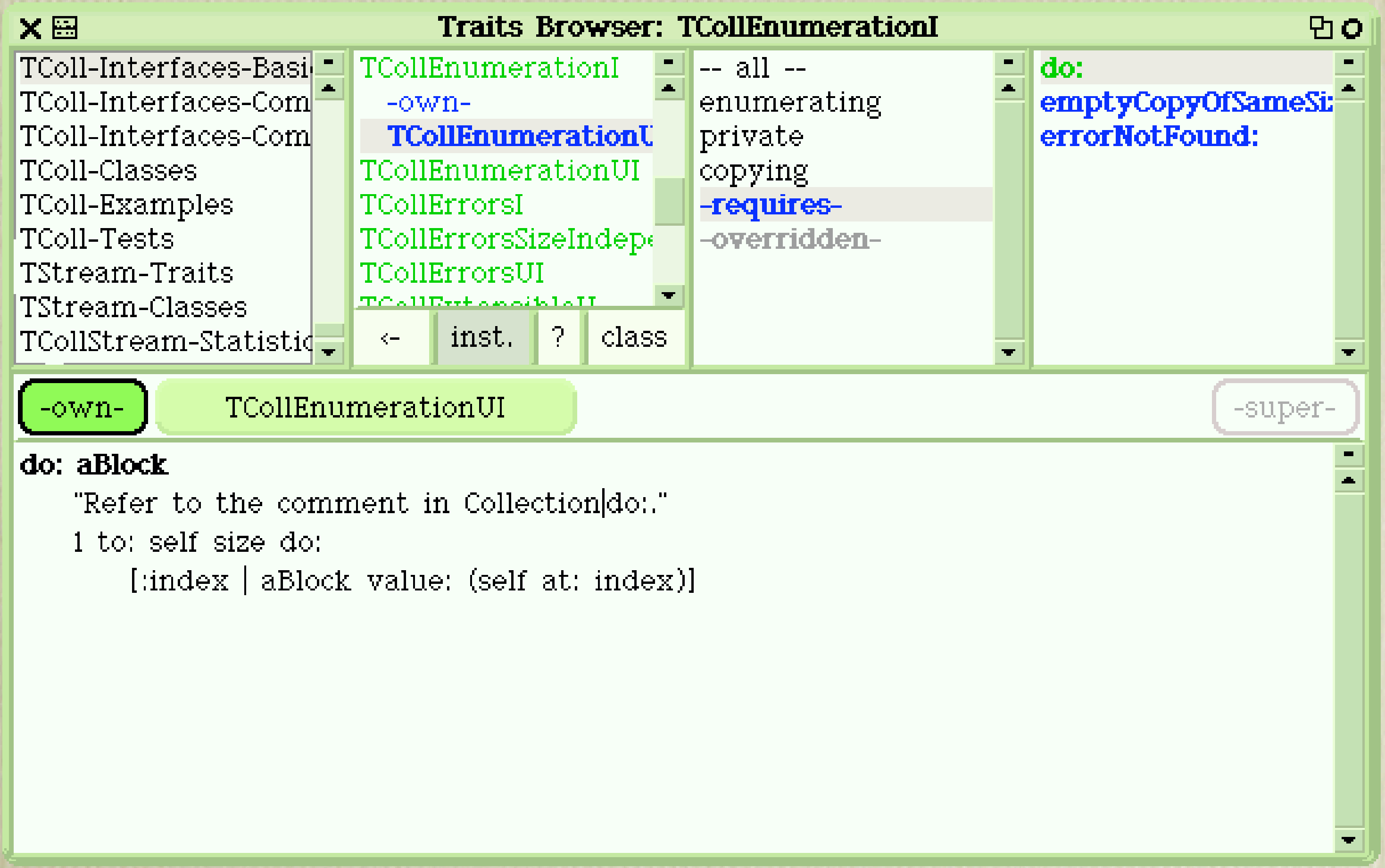
```
errorNotFound: index
    self requirement
```

Enumerations in the traits browser



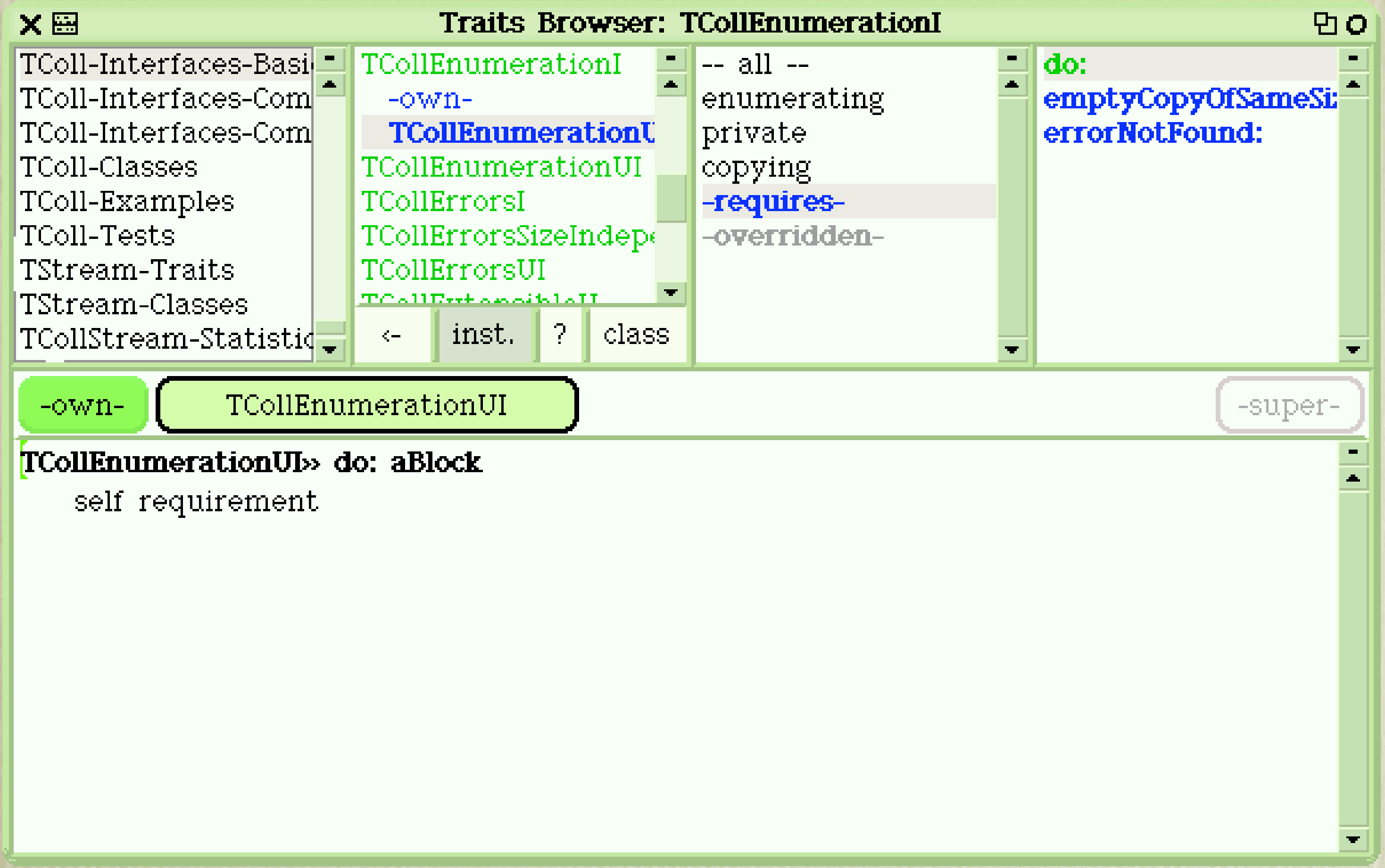
```
do: aBlock
	"Refer to the comment in Collection|do:."
	1 to: self size do:
		[:index | aBlock value: (self at: index)]
```

Enumerations in the traits browser



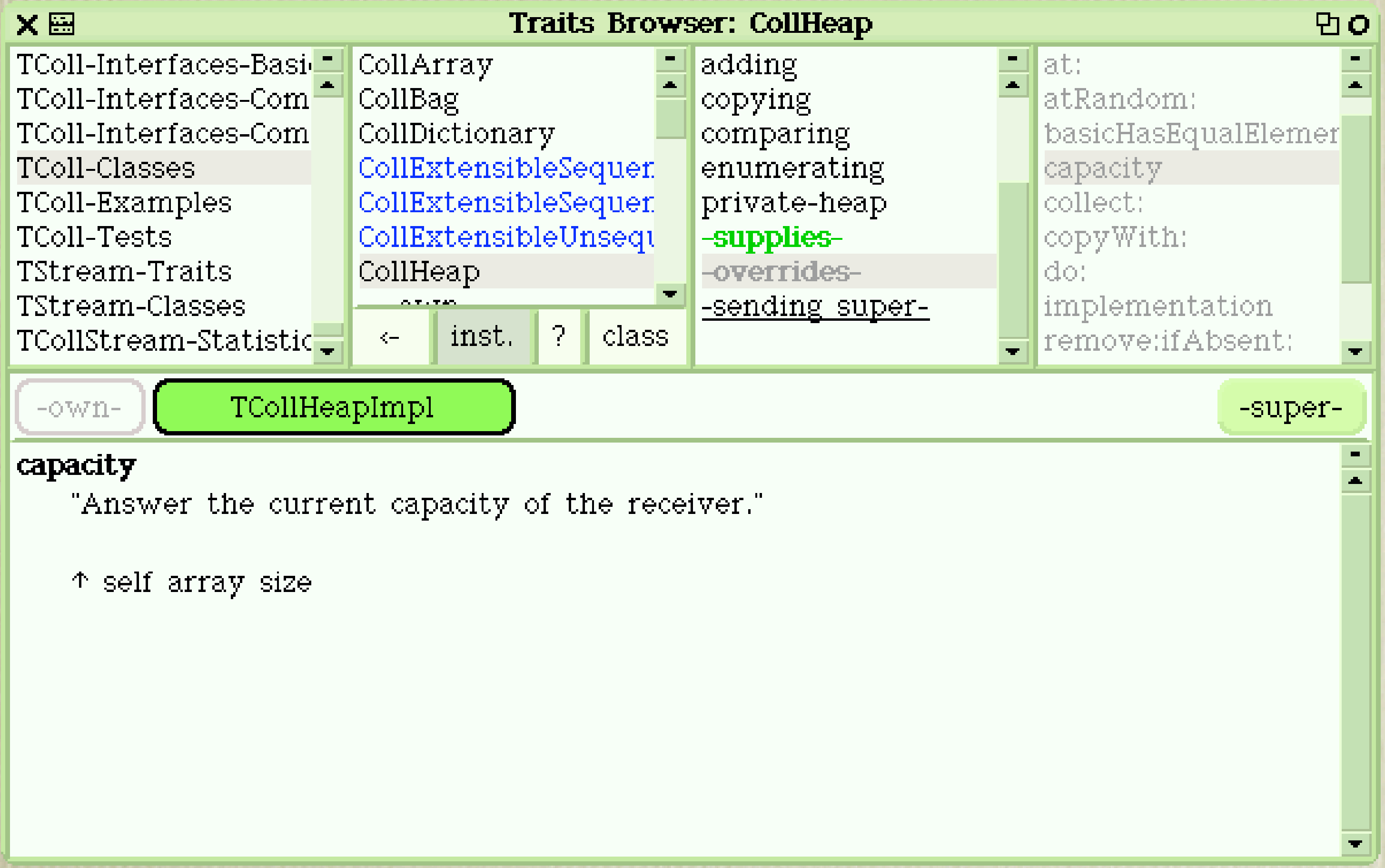
```
do: aBlock
	"Refer to the comment in Collection|do:."
	1 to: self size do:
		[:index | aBlock value: (self at: index)]
```

Enumerations in the traits browser



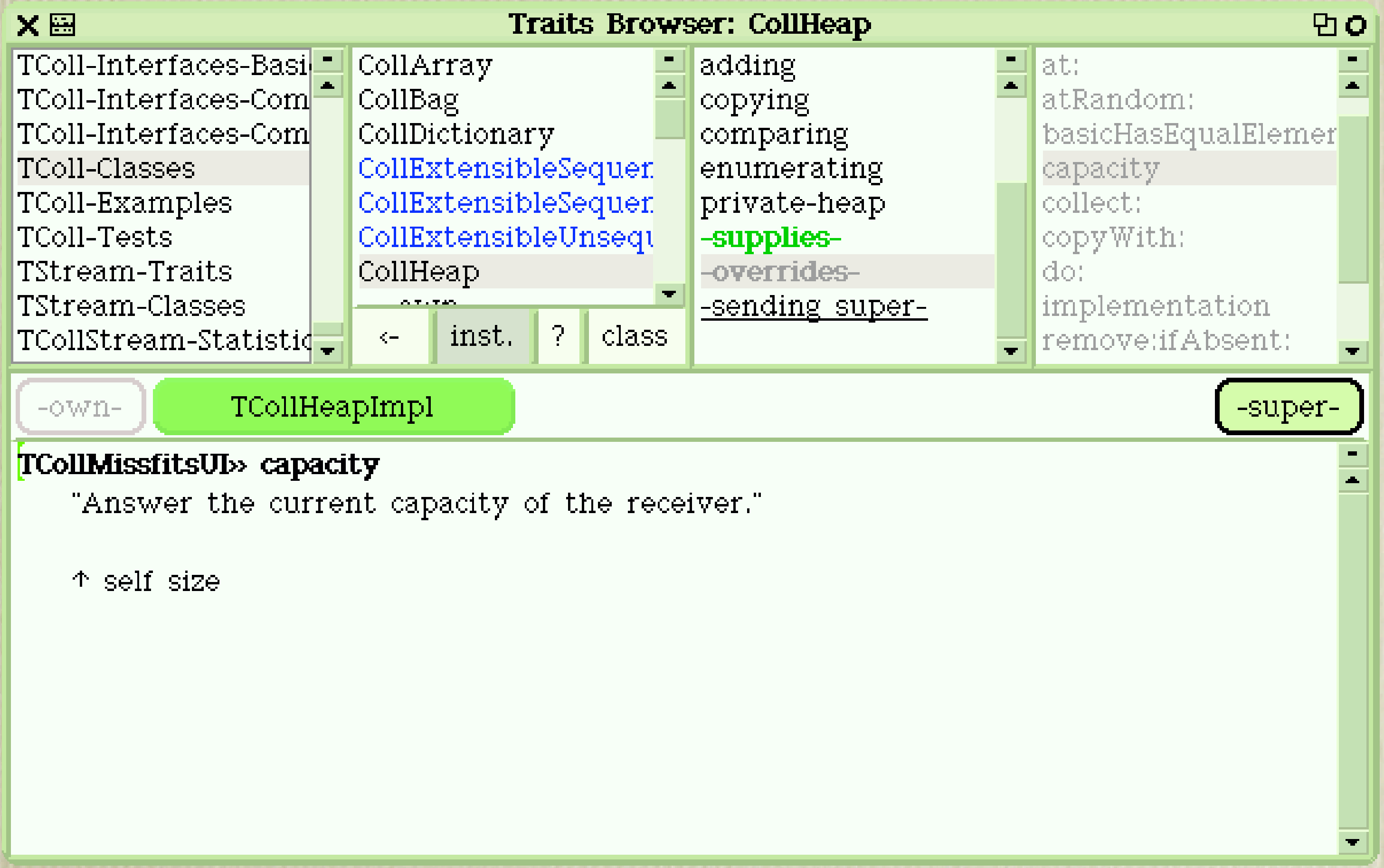
```
TCollEnumerationUI» do: aBlock
    self requirement
```

overrides virtual category



```
capacity
	"Answer the current capacity of the receiver."

	↑ self array size
```

overrides virtual category

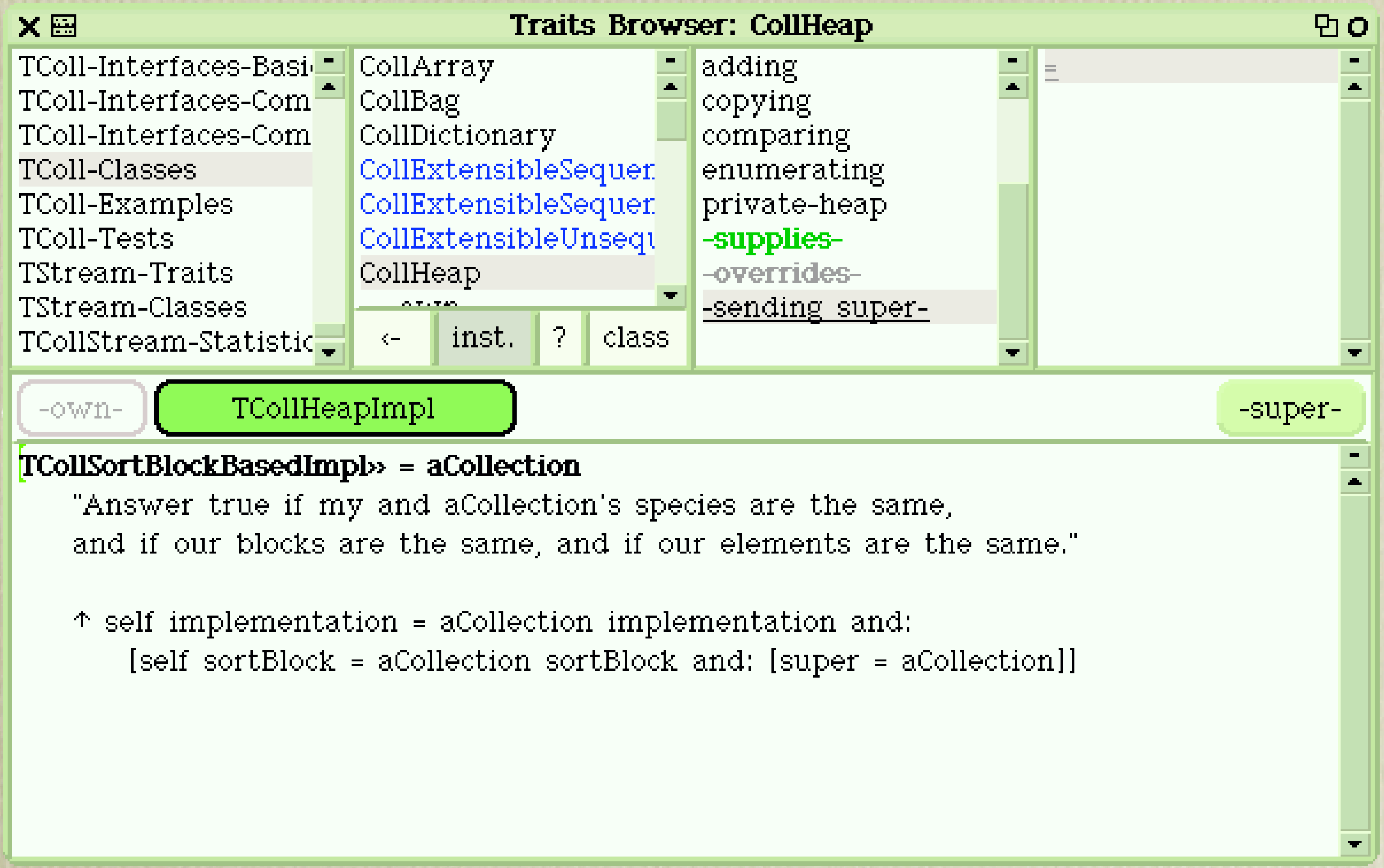


```
TCollMissfitsUI» capacity
    "Answer the current capacity of the receiver."

    ↑ self size
```

sending-super virtual category

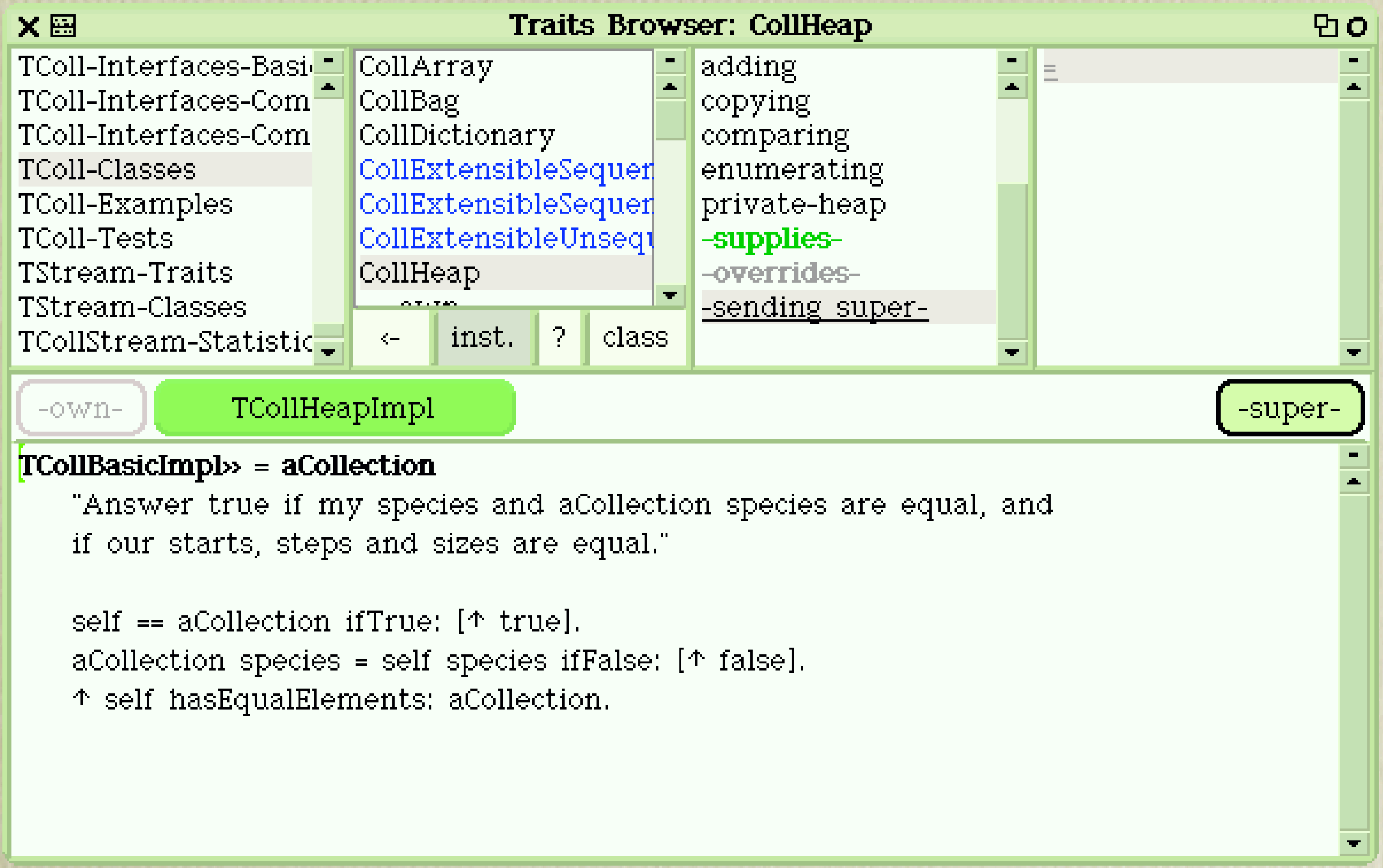
- Contains all the methods in this class or trait that make `super`-sends

sending-super virtual category



```
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

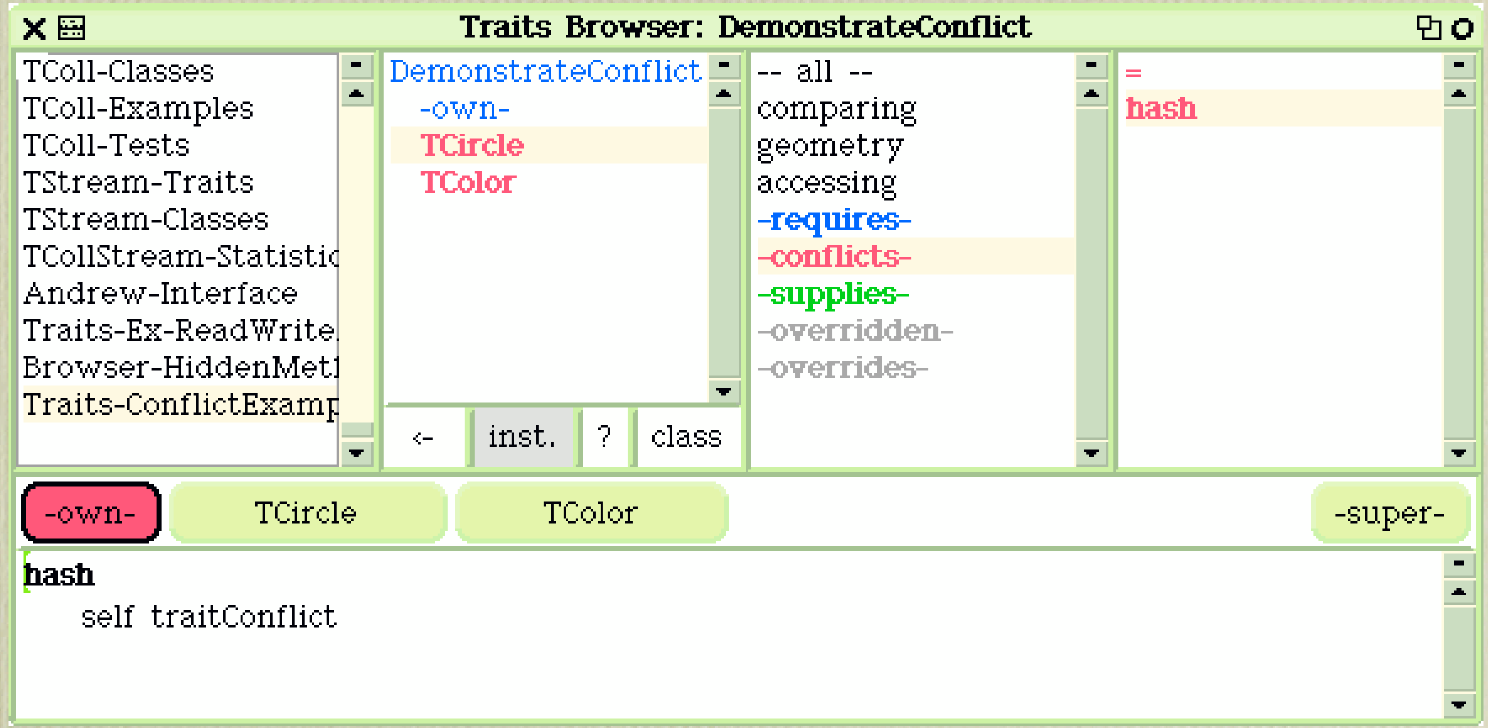


```
TCollBasicImpl>> = aCollection
    "Answer true if my species and aCollection species are equal, and
    if our starts, steps and sizes are equal."

    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


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

```
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 □ (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

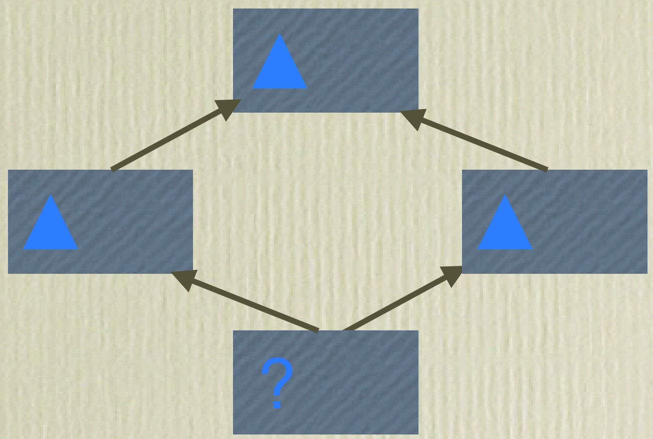
- XP and trait programming share practices
  - continuous design
  - refactoring
  - testing
  - pair programming
  - collective ownership

Tools and methodology interact

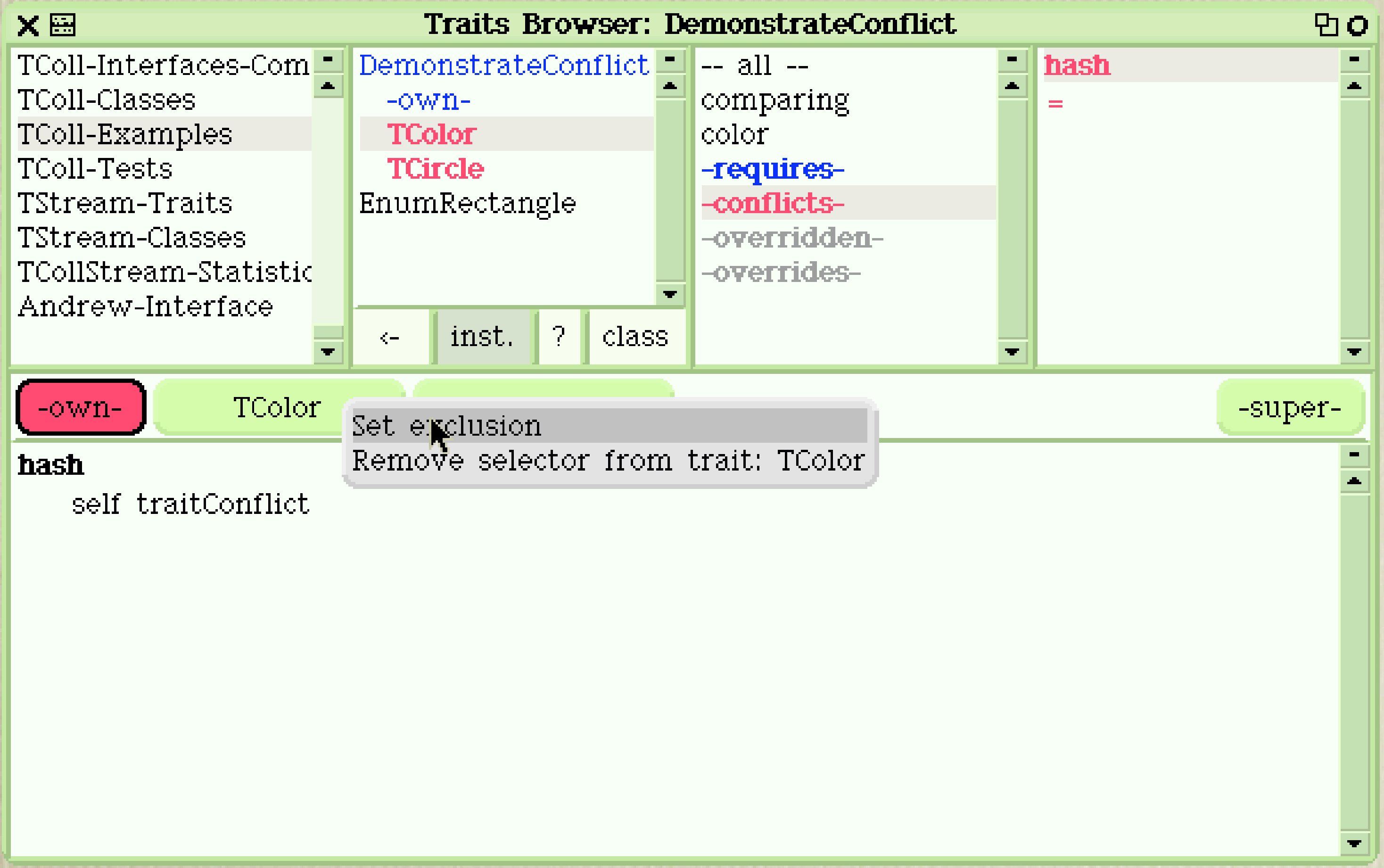
- Methodology without tool support □ pious hope
- Tools without methodology □ too much rope
- Trait language features and browser co-evolved with the methodology

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

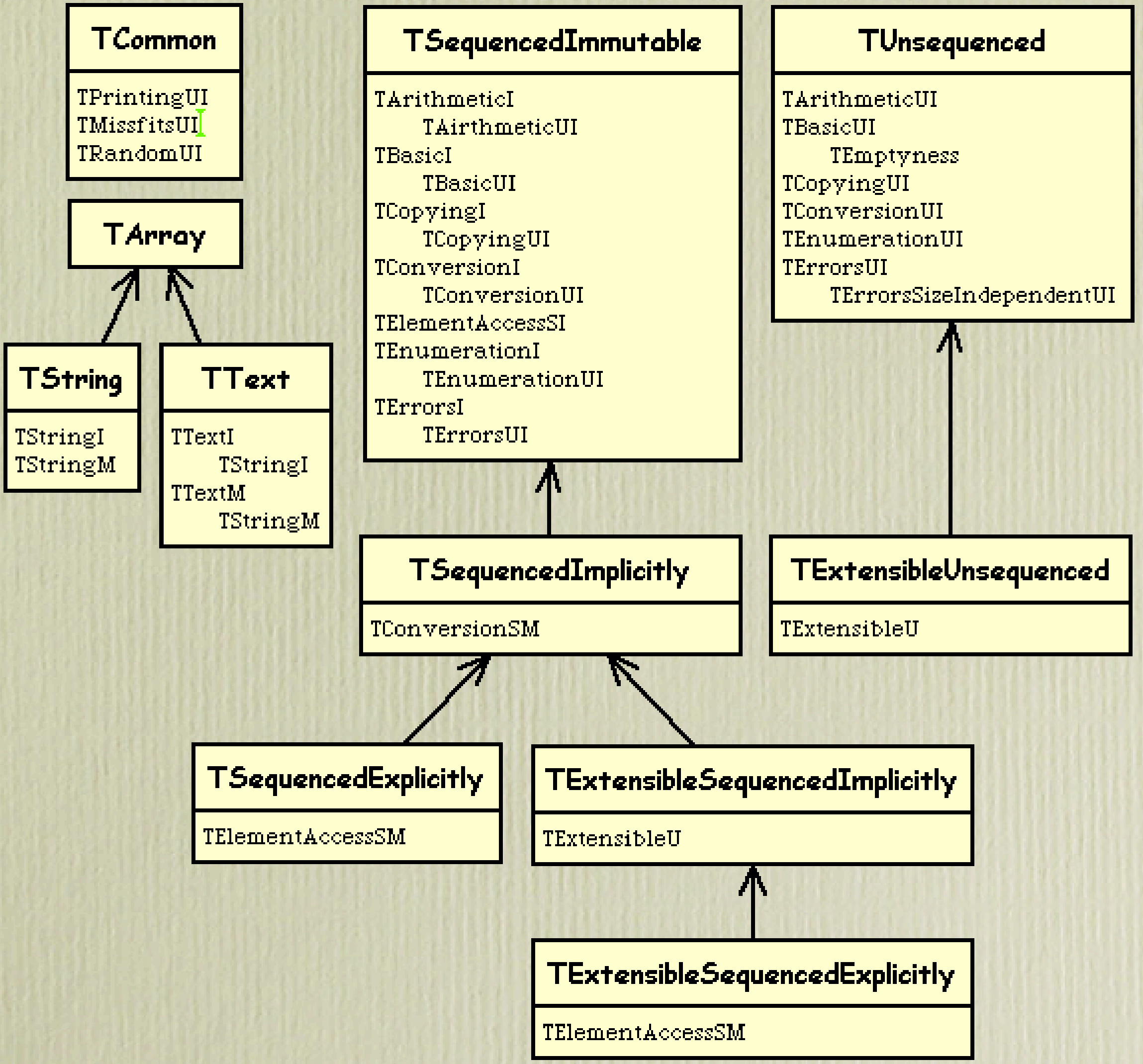


```
hash
    self traitConflict
```

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 (1/2)

- 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