


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?

- ① 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

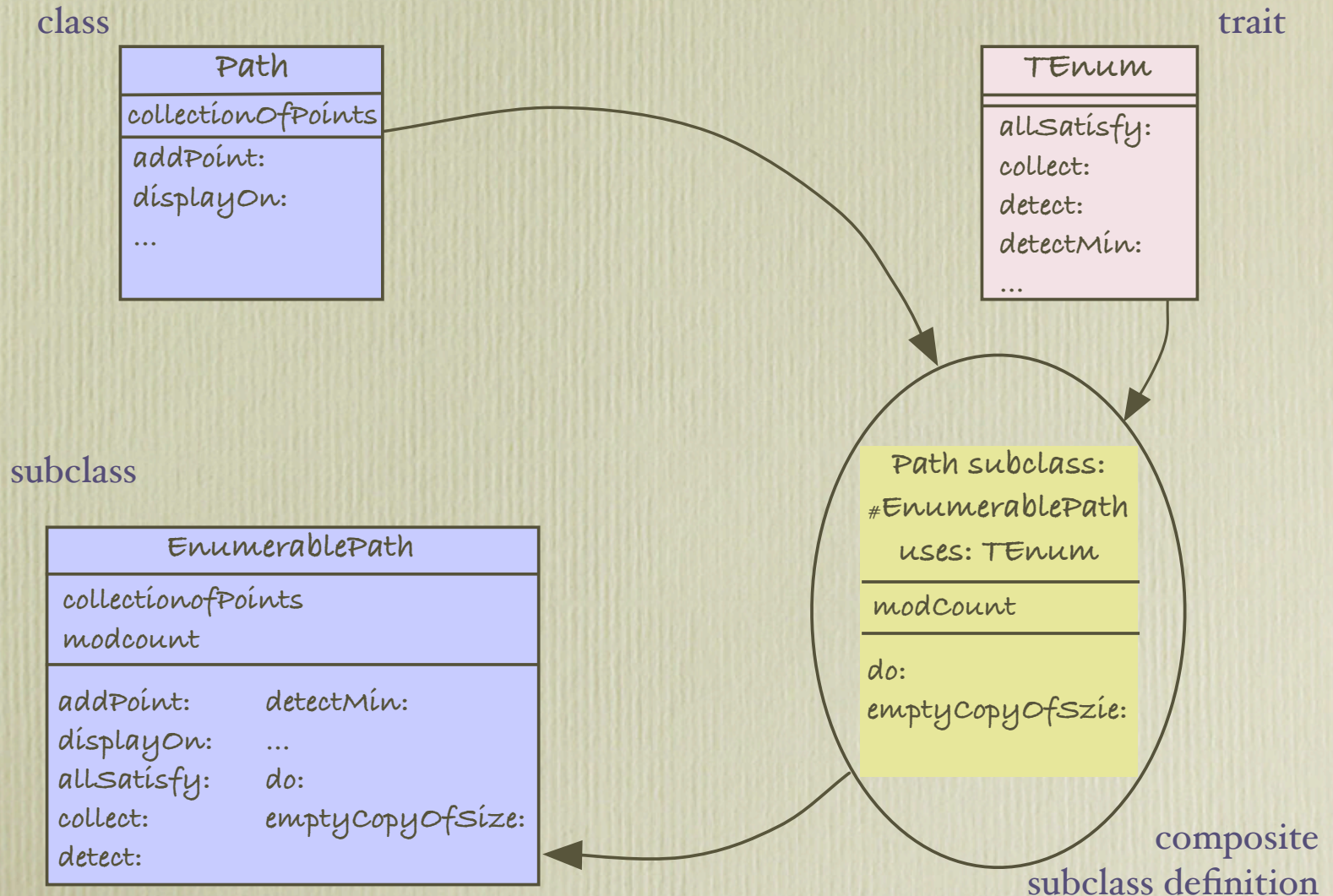
What about Path?

- ⦿ 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

- ④ Smalltalk is a dynamically typed, class-based language with single inheritance
- ④ 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

- ④ How does the programmer manipulate traits ?
 - Tools — the trait browser

Today's talk: two questions

- ④ How does the programmer manipulate traits ?
 - Tools — the trait browser
- ④ How do traits change the way that programs are written?

Today's talk: two questions

- ① How does the programmer manipulate traits ?
 - Tools — the trait browser
- ① 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 window titled "Traits Browser: TCollEnumerationUI". The left sidebar lists various trait categories, and the main area displays the trait hierarchy for TCollEnumerationUI. The trait is expanded to show its methods, with "select:" highlighted. Below the trait list, the implementation of "select: aBlock" is shown, including a description and the corresponding Smalltalk code.

Traits Browser: TCollEnumerationUI

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

TCollEnumerationUI

- own-
- TCollErrorsI
- TCollErrorsSizeIndependent
- TCollErrorsUI
- TCollExtensibleU
- TCollMissfitsUI
- TCollPrimitiveUI

-- 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 window titled "Traits Browser: TCollEnumerationUI". The left sidebar lists various trait categories, and the main area displays the trait definition for TCollEnumerationUI. The trait includes several methods: inject:into:, intersection:, noneSatisfy:, reject:, select:, select:thenCollect:, union:, and withIndexDo: (highlighted). The withIndexDo method is defined as follows:

```
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 window is divided into several panes:

- Left Pane:** A tree view of traits. The selected trait is "TCollEnumerationUI". Other visible traits include "TColl-Interfaces-Basi...", "TColl-Interfaces-Com...", "TColl-Interfaces-Com...", "TColl-Classes", "TColl-Examples", "TColl-Tests", "TStream-Traits", "TStream-Classes", and "TCollStream-Statistic".
- Top Middle Pane:** A list of trait methods and symbols. The selected item is "do:". Other items include "difference:", "do:separatedBy:", "do:without:", "doWithIndex:", "emptyCopyOfSameSi...", "errorNotFound:", "groupBy:having:", and "inject:into:". There are also symbols like "-- all --", "enumerating", "converting", "private", "copying", and "-requires-".
- Bottom Middle Pane:** A navigation bar with buttons for "<-", "inst.", "?", and "class".
- Bottom Right Pane:** A navigation bar with buttons for "-own-" and "-super-".
- Main Content Area:** The selected trait's definition is shown as a block of code:

```
do: aBlock
  self requirement
```

Enumerations in the traits browser

The screenshot shows the Traits Browser window titled "Traits Browser: TCollEnumerationUI". The left sidebar lists various trait categories, with "TColl-EnumerationUI" selected. The main pane displays the trait's definition, including a list of methods: "enumerating", "converting", "private", "copying", and "-requires-". The "-requires-" method is highlighted. Below the main pane, there are two buttons: "-own-" and "-super-". The bottom section of the browser shows the "errorNotFound: index" method with a "self requirement".

```
TColl-EnumerationUI -- all --  
-own- enumerating  
TCollErrorsI converting  
TCollErrorsSizeIndep private  
TCollErrorsUI copying  
TCollExtensibleU -requires-  
TCollMissfitsUI  
TCollPrimitiveUI  
<- inst. ? class  
do:  
emptyCopyOfSameSi  
errorNotFound:  
-own- -super-  
errorNotFound: index  
self requirement
```


Enumerations in the traits browser

The screenshot shows the Traits Browser window for `TCollEnumerationI`. The left pane shows a tree view of traits, with `TCollEnumerationI` selected. The middle pane shows the trait's methods and annotations, including `enumerating`, `private`, `accessing`, `error handling`, `copying`, `-requires-`, `-supplies-`, and `-overrides-`. The right pane shows the trait's methods, with `do:` selected. Below the panes, a button labeled `-own-` is active, and the trait `TCollEnumerationUI` is selected. The bottom pane shows the implementation of `do: aBlock` for `TCollEnumerationUI`.

```
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

The screenshot shows the Traits Browser window for `TCollEnumerationI`. The left sidebar lists various trait categories, with `TColl-Interfaces-Basic` selected. The main pane displays the trait definition for `TCollEnumerationI`, including its methods and annotations. The `do:` block is highlighted, showing the implementation of the `aBlock` method. The `TCollEnumerationUI` trait is also visible in the list, indicating it implements `TCollEnumerationI`.

```
TCollEnumerationI
-- all --
enumerating
private
copying
-requires-
-overridden-

do:
emptyCopyOfSameSi
errorNotFound:
```

`TCollEnumerationUI`

```
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

The screenshot shows the Traits Browser interface for `TCollEnumerationI`. The left sidebar lists various trait categories, with `TColl-Interfaces-Com` selected. The main area displays the hierarchy of traits, including `TCollEnumerationI`, `TCollEnumerationUI`, `TCollErrorsI`, `TCollErrorsSizeIndep`, `TCollErrorsUI`, and `TCollExtensibleUI`. The `TCollEnumerationUI` trait is highlighted. Below the trait list, there are buttons for `-own-`, `TCollEnumerationUI`, and `-super-`. The bottom panel shows the `do: aBlock` method for `TCollEnumerationUI`, with the implementation `self requirement`.

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

TCollEnumerationI
-own-
TCollEnumerationUI
TCollEnumerationUI
TCollErrorsI
TCollErrorsSizeIndep
TCollErrorsUI
TCollExtensibleUI

-- all --
enumerating
private
copying
-requires-
-overridden-

do:
emptyCopyOfSameSi
errorNotFound:

TCollEnumerationUI
TCollEnumerationUI

TCollEnumerationUI>> do: aBlock
self requirement
```

overrides virtual category

The screenshot shows the Traits Browser for the `CollHeap` trait. The browser is divided into several panes:

- Left Pane:** A tree view of traits. `TColl-Classes` is selected, and `CollHeap` is highlighted in the list.
- Middle Pane:** A list of methods and categories for `CollHeap`. The `-supplies-` and `-overrides-` categories are expanded, showing `capacity` under `-supplies-` and `private-heap` under `-overrides-`.
- Right Pane:** A list of methods for `CollHeap`, including `at:`, `atRandom:`, `basicHasEqualElementer`, `capacity`, `collect:`, `copyWith:`, `do:`, `implementation`, and `remove;ifAbsent:`.
- Bottom Pane:** The implementation of the `capacity` method in `TCollHeapImpl`. The method is defined as:

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

  ↑ self array size
```


overrides virtual category

The screenshot shows the Traits Browser for the `CollHeap` trait. The browser is divided into several panes:

- Left Pane:** A list of trait categories including `TColl-Interfaces-Basic`, `TColl-Interfaces-Common`, `TColl-Interfaces-Common`, `TColl-Classes`, `TColl-Examples`, `TColl-Tests`, `TStream-Traits`, `TStream-Classes`, and `TCollStream-Statistics`.
- Middle Pane:** A list of traits including `CollArray`, `CollBag`, `CollDictionary`, `CollExtensibleSequencer`, `CollExtensibleSequencer`, `CollExtensibleUnsequencer`, and `CollHeap`. Below this list are buttons for `<-`, `inst.`, `?`, and `class`.
- Right Pane:** A list of methods including `adding`, `copying`, `comparing`, `enumerating`, `private-heap`, `-supplies-`, `-overrides-`, and `-sending super-`.
- Bottom Pane:** A list of methods including `at:`, `atRandom:`, `basicHasEqualElement`, `capacity`, `collect:`, `copyWith:`, `do:`, `implementation`, and `remove;ifAbsent:`.

Below the panes, there are buttons for `-own-`, `TCollHeapImpl` (highlighted), and `-super-`.

The main content area shows the following text:

```
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 the Traits Browser for the `CollHeap` class. The browser is divided into several panes:

- Left Pane:** A tree view of the class hierarchy. The selected path is `TColl-Interfaces-Basic` > `CollHeap`.
- Middle Pane:** A list of methods and traits for `CollHeap`. The selected item is `-sending super-`.
- Right Pane:** A list of traits for `CollHeap`. The selected trait is `=`.
- Bottom Pane:** A detailed view of the `-sending super-` trait. It shows the implementation for `TCollSortBlockBasedImpl` as `aCollection`. The implementation text is: "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." Below this, it shows the self implementation: `↑ self implementation = aCollection implementation and: [self sortBlock = aCollection sortBlock and: [super = aCollection]]`.

sending-super virtual category

The screenshot shows the Traits Browser interface for the `CollHeap` trait. The left pane lists various trait categories, and the middle pane shows the trait hierarchy. The right pane lists methods, with `-sending super-` highlighted. Below the browser, the implementation `TCollHeapImpl` is shown, including its superclass `TCollBasicImpl` and a description of its behavior.

Traits Browser: CollHeap

Left pane: TColl-Interfaces-Basi, TColl-Interfaces-Com, TColl-Interfaces-Com, TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, TCollStream-Statistic

Middle pane: CollArray, CollBag, CollDictionary, CollExtensibleSequer, CollExtensibleSequer, CollExtensibleUnsequ, CollHeap

Right pane: adding, copying, comparing, enumerating, private-heap, **-supplies-**, -overrides-, **-sending super-**

Buttons: -own- **TCollHeapImpl** -super-

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

The screenshot shows the Traits Browser application window titled "Traits Browser: DemonstrateConflict". The interface is divided into several panes:

- Left Pane:** A list of project folders including TColl-Classes, TColl-Examples, TColl-Tests, TStream-Traits, TStream-Classes, TCollStream-Statistic, Andrew-Interface, Traits-Ex-ReadWrite, Browser-HiddenMet, and Traits-ConflictExamp.
- Top Middle Pane:** Shows the selected trait "DemonstrateConflict" with its sub-traits: "-own-", "TCircle", and "TColor".
- Top Right Pane:** Lists various trait relationships: "-- all --", "comparing", "geometry", "accessing", "-requires-", "-conflicts-", "-supplies-", "-overridden-", and "-overrides-".
- Bottom Middle Pane:** A navigation bar with buttons for "-own-", "TCircle", "TColor", and "-super-".
- Bottom Right Pane:** Displays the "hash" trait with the implementation "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

- ④ Traits support modularization directly (3)
- ④ Trait methods can be reused anywhere in a hierarchy (4)
- ④ Inheritance with traits allows reuse of the \square (5)
- ④ 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

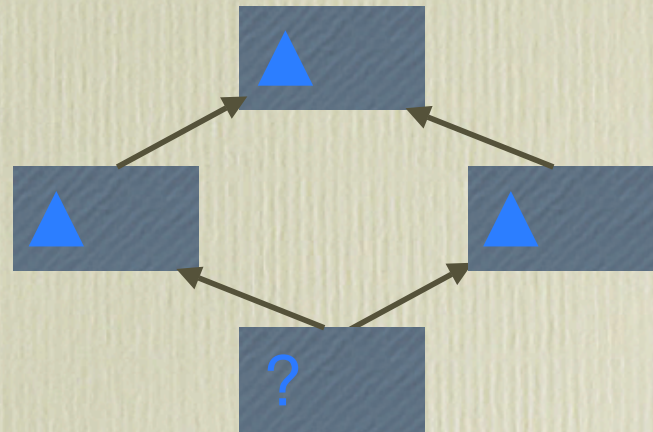
- XP and trait programming share practices
 - continuous design
 - refactoring
 - testing
 - pair programming
 - collective ownership

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 window titled "DemonstrateConflict". The left sidebar lists various trait collections, with "TColl-Examples" selected. The main area displays the "DemonstrateConflict" trait, which includes a list of other traits: **TColor**, **TCircle**, and EnumRectangle. Below this list are buttons for navigation: "<", "inst.", "?", and "class".

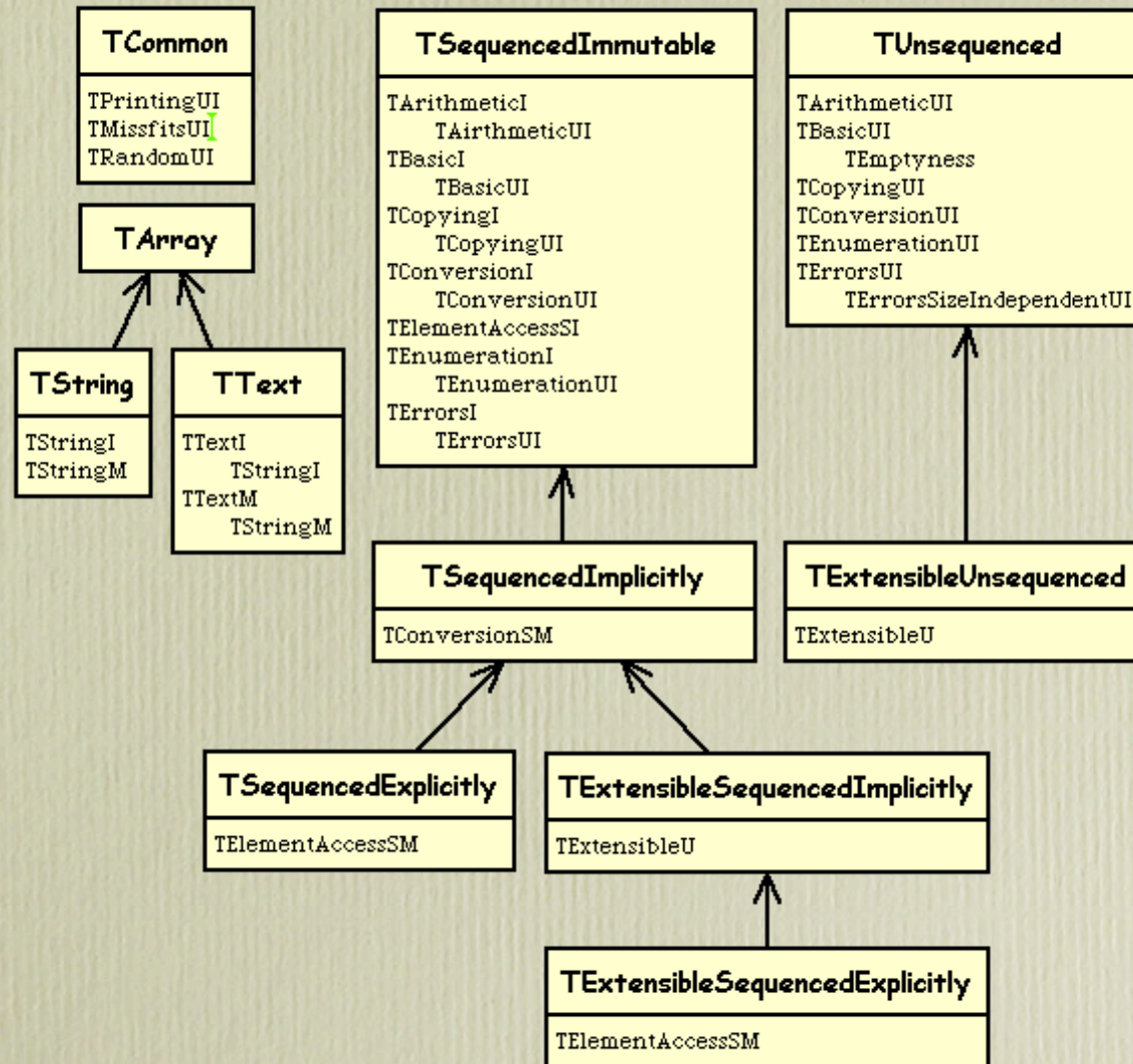
Below the main area, there are buttons for "-own-", "TColor", and "-super-". A context menu is open over the "TColor" button, showing two options: "Set exclusion" and "Remove selector from trait: TColor".

The right sidebar shows the "hash" trait, which has a value of "=". Below this, the "hash" trait is expanded to show the code: "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