Featherweight Traits

Oscar Nierstrasz
Stéphane Ducasse
Nathanael Schärli

Motivation

How to implement traits in statically-typed languages?

i.e., Java & C#

Featherweight Java

- Purely functional object calculus
  - ☞ *Rules for: syntax, subtyping, field/method/type lookup, expression/method/class type-checking, computation*
- Captures minimal aspects of Java types
  - ☞ *Ignores side-effects, super-sends, overloaded methods, reflection, concurrency etc.*
- Designed to answer the question:
  - ☞ *How can Java be extended with generics without breaking the type system?*

A perfect basis for studying traits in Java-like languages!

Featherweight Traits

- Start with FJ
  - ☞ *Consider only minimal changes*
- Adopt the principle:
  - ☞ *"A trait is kinda, sorta like a class"*
- Take care for:
  - ☞ *Conflicts*
  - ☞ *Required methods*
- Ignore:
  - ☞ *Aliasing, exclusion, super-sends*

FT Syntax

```
class C extends C uses T̄ { ... }

trait T uses T̄ {M̄}
```

Subtyping

$$\frac{\texttt{class C extends D uses } \bar{\texttt{T}}\texttt{\{ ... \}}}{\texttt{C<:D} \quad \forall i.\texttt{C<:T}_i}$$

$$\frac{\texttt{trait T uses } \bar{\texttt{T}}\ \texttt{\{}\bar{\texttt{M}}\texttt{\}}}{\forall i.\texttt{T<:T}_i}$$

Auxiliary functions

- Field lookup
  - ☞*not needed*
- Method type lookup
  - ☞*Give priority to class, then traits, then superclass*
- Method body lookup
  - ☞*Similar*

Next Steps …

- Expression typing
  - ☞ *Unchanged (!)*
- Method typing
  - ☞ *Need to account for conflicts (no type inconsistencies allowed)*
- Class typing
  - ☞ *Need to check that conflicts are resolved and required methods provided*
- Reductions rules
  - ☞ *Unchanged! (due to flattening)*

- Generics
  - ☞ *Generic traits are like generic classes …*