Various predefined “variables”:

* super (a *function* with the same parameters as this:method)
* this (i.e. “this object”)
* this:public / this:protected / this:private
* this:module
* this:package
* this:class
* this:property (only w/in a method w/in a property)
* this:method
* this:frame

Annotation on:

* Class
* Property
* Method
* Parameter
* Variable

Property

* Type Specifier
* Name

Type Specifier is one of:

* Qualified Type
* Type Definition
* Type Reference
* Type Alias

Type Definition is one of:

* A set of Methods and Properties
* Union of two Type Specifiers
* Intersection of two Type Specifiers
* Complement (i.e. difference) of two Type Specifiers

Type Reference is one of:

* this:class
* Outer class of a Type Reference
* Inner class (by *Name*) of a Type Reference
* Type Parameter (by *Name* or by *Index*) of a Type Reference
* Type Parameter (by *Name* or by *Index*) of a Method Reference
* Return Type

Method Reference is one of:

* this:method
* TODO

Property Reference is one of:

* this:property
* Property (by *Name*) of a Type Reference
* Property (by *Name*) of a Method Reference
* TODO

Type alias constant is:

* Name
* Type specifier

Also need a way to say “this class” in a virtual sense, or things relative to “this class”. For example, think about a StringBuilder … it often returns StringBuilder. But if you extend it to create a SuperStringBuilder, then you would want those same methods to return the type SuperStringBuilder (so as not to lose the additional aspects that might be available in its public interface). So just like one can define a parameterized type using a <T>, there must also be some predefined “T” that means “the type of the *this*”. Similarly, it must be possible to return types nested within “the type of the *this*”, so if Map returns a Map.Entry, then SuperMap would return a SuperMap.Entry.

class Builder {

public Builder addX(int x);

public class InnerBuilder { … }

public InnerBuilder createInnerBuilder();

}

class BetterBuilder extends Builder {

// how to make these three implicit?!?

public BetterBuilder addX(int x);

public class InnerBuilder extends Builder.InnerBuilder { … }

public BetterBuilder.InnerBuilder createInnerBuilder();

public BetterBuilder addY(int y);

}

How about:

class Builder {

public this:class addX(int x);

public class InnerBuilder { … }

public InnerBuilder createInnerBuilder(); // implicitly this:class.InnerBuilder

}

class BetterBuilder extends Builder {

@override class InnerBuilder {…}

public this:class addY(int y);

}

Or what if it were entirely implicit, i.e. specifying the name of “this class” would implicitly mean “this:class”.

class Builder {

public Builder addX(int x); // implicitly this:class return type

public class InnerBuilder { … }

public InnerBuilder createInnerBuilder(); // implicitly this:class.InnerBuilder

}

class BetterBuilder extends Builder {

@override class InnerBuilder {…}

public BetterBuilder addY(int y);

}

--

class Variable<T> {

T get();

void set(T value);

String Name;

}

Traits for variables (including local variables, parameters, and properties) could just be a Trait<Variable>

Mixin as a stateless trait?

--