1 An Opening Example

Featherweight Java is a core subset of Java that allows us to reason about an object-oriented language using a higher level model than our previous model based on the λ-calculus. In Featherweight Java, objects are the basic building blocks with which we deal, as opposed to objects being an abstraction of lower-level components. The discussion started with the following brief example:

```java
class A extends Object { A() { super(); } }
class B extends Object { B() { super(); } }
class Pair extends Object {
    Object fst;
    Object snd;
    Pair(Object fst, Object snd) { super; this.fst = fst; this.snd = snd; }
    Pair setfst(Object newfst) { return new Pair(newfst, this.snd); }
}
```

The example appears very close to Java code with a few notable differences. In the course of discussion, the following differences were noted:

- The classes `A`, `B`, and `Pair` all explicitly state that they extend `Object`.
- The `setfst` method returns a new object instead of doing an assignment.
- The explicit constructors that just called `super`.
- There is strict parity between the constructor arguments and the class member variables.

With these items in mind, we then outlined the syntax for Featherweight Java as follows:

```plaintext
CL ::= class C extends C { C ċ; K Č M}  class decl.
K ::= C ċ ě super (č); this.č = ě;  constructor
M ::= C m(Č x) return t;  method decl.
t ::= x  variable
t.č  field access
t.m(č)  method invocation
new C(č)  object creation
(C) t  cast
v ::= new C(ν)  object creation
```

One link to our previous work is that the value in the λ-calculus was λx.t and here our value is the object creation.
2 Evaluation and Sub-typing Rules

In the following evaluation rules we explicitly pass the program text, \( P \), to each rule.

\[
\begin{align*}
P \vdash t \rightarrow t' \\
\hline
P \vdash t.f \rightarrow t'.f \\
\end{align*}
\]

\[
\begin{align*}
P \vdash t \rightarrow t' \\
\hline
P \vdash t.m(t) \rightarrow t'.m(t) \\
\end{align*}
\]

\[
\begin{align*}
P \vdash t \rightarrow t' \\
\hline
P \vdash (C)t \rightarrow (C)t' \\
\end{align*}
\]

\[
\begin{align*}
P \vdash (C)v \rightarrow (C)v \\
\end{align*}
\]

\[
\begin{align*}
P \vdash t_i \rightarrow t'_i \\
\hline
P \vdash v.m(\bar{v}, \ldots, t_i, \ldots, t_n) \rightarrow v.m(\bar{v}, \ldots, t'_i, \ldots, t_n) \\
\end{align*}
\]

\[
\begin{align*}
P \vdash t_i \rightarrow t'_i \\
\hline
P \vdash new\ C(\bar{v}, \ldots, t_i, \ldots, t_n) \rightarrow new\ C(\bar{v}, \ldots, t'_i, \ldots, t_n) \\
\end{align*}
\]

\[
\begin{align*}
P \vdash D m(\bar{E}x)\{\text{return } t; \} \in \text{methods}(P,C) \\
\hline
P \vdash new\ C(\bar{v})\cdot m(\bar{u}) \rightarrow [\bar{x} \mapsto \bar{u}, this \mapsto new\ C(\bar{v})]t \\
\end{align*}
\]

\[
\begin{align*}
P \vdash fields(P,C) = D_1e_1, \ldots, D_ne_n \\
\hline
P \vdash new\ C(\bar{v}_1, \ldots, \bar{v}_n)\cdot f \rightarrow \bar{v}_i \\
\end{align*}
\]

\[
\begin{align*}
P \vdash D <: C \\
\hline
P \vdash (C)\cdot new\ D.(\bar{v}) \rightarrow new\ D(\bar{v}) \\
\end{align*}
\]

\[
\begin{align*}
P \vdash A <: B \\
B <: C \\
\hline
P \vdash A <: C \\
\end{align*}
\]

\[
\begin{align*}
P \vdash A <: A \\
\end{align*}
\]

\[
\begin{align*}
class\ A\ extends\ B \ldots \in\ P \\
A <: B \\
\end{align*}
\]

\[
\begin{align*}
P \vdash B <: C \\
class\ A\ extends\ B \ldots \in\ P \\
A <: B \\
\end{align*}
\]
3 Nominal versus Structural Sub-typing

With the evaluation rules complete, our attention turned to nominal versus structural sub-typing. A lively discussion ensued with the following points made both pro and con for nominal sub-typing.

+ Simple for the programmer
+ Simple for the compiler
+ No spurious sub-types, although it’s pointed out structuralists rebuttal is the use of wrappers in the form of single alternative variants.
- Polymorphism is more difficult
- Less flexible
+ “No-brainer choice for global variant tags”
- You have to check for cycles in the sub-types
+ Documents the programmer’s intent
- Disconnected development

4 From Academic Ideas to Common Practice

A brief distraction came in the form of a discussion on the adoption of academic ideas into the realm of common practice. One possible trend that was mentioned was the increasing rate at which ideas are adopted. To this end, Professor Flanagan took a moment to give a plug for Eric S. Raymond’s book, “The Cathedral and the Bazaar”.

5 Type Rules

We concluded by coming up with a set of type rules as follows:

\[
\frac{x : C \in \Gamma}{P, \Gamma \vdash x : C}
\]

\[
\frac{P, \Gamma \vdash t : C \quad C \prec T}{P, \Gamma \vdash (T)t : T}
\]

\[
\frac{P, \Gamma \vdash t : C \quad \text{fields}(P, C) = D_1f_1, \ldots, D_nf_n}{P, \Gamma \vdash t.f : D_i}
\]

\[
\frac{P, \Gamma \vdash t : C \quad D m(\bar{C} \bar{x})\{\ldots\} \in \text{methods}(P, C) \quad P, \Gamma \vdash \bar{t} : \bar{C}}{P, \Gamma \vdash t.m(\bar{t}) : D}
\]

\[
\frac{P, \Gamma \vdash \bar{t} : \bar{D} \quad \text{fields}(P, C) = \bar{D}\bar{e}}{P, \Gamma \vdash \text{new } C(t) : C}
\]
\( D \in \text{ClassNames}(P) \quad \forall f_i \in \vec{f}, f_i \in \text{fields}(P, D) \)

fields\((P, D)\) is OK \hspace{1cm} \( \bar{D}g = \text{fields}(P, D) \)

\( P \vdash \text{class } C \text{ extends } D \)

\( \bar{f} \in \text{fields}(P, D) \)

\( \bar{K} = C(\bar{D}g, \bar{C}f) \{ \text{super}(\bar{g}; \text{this.} \bar{f} = f) \} \)

\( \bar{K} \bar{m} \) is OK

\[ \forall CL \in P, P \vdash CL \text{ is OK} \quad \text{ClassesOnce}(P) \]

\[ P \text{ is OK} \]

\[ \text{if } \text{Em}(\bar{E}z)\{ \ldots \} \in \text{methods}(P, D) \text{ then } E = D \text{ and } \bar{C} = \bar{E} \]

class \( C \text{ extends } C' \{ \ldots \} \in P \quad \text{Override OK}(C', D, m, \bar{C}, P) \quad \text{this : } C, \bar{x} : \bar{c} \quad P \vdash t : D \)

\( P \vdash D m(\bar{C}f)\{ \text{return}; \} \) is OK in \( C \)