

Featherweight Java (FJ)

Authors: A. Igarashi, B. Pierce, P. Wadler, appeared in OOPSLA 1999.

Tiny subset of Java that models subtyping and inheritance.

- ◆ Includes core OO features but no concurrency/reflection/inner classes.
- ◆ No assignment, interfaces, overloading, super calls, null pointers.
- ◆ Has object creation, method invocation, field access, cast, variables, recursive class declarations, subtyping

Example (Class Declarations)

```
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(); self.fst := fst; self.snd := snd;}
    Pair setfst(Object newfst) {
        //method bodies are 'return expressions'
        return new Pair(newfst, self.snd);}}}
```

Class Table, [CT](#), maps class names to class declarations.

An FJ program is ([CT,t](#)).

FJ program (CT, t)

Say t is

```
new Pair(new A(), new B()).setfst(new B())
```

Then: t evaluates to `new Pair(new B(), new B())`

FJ program (CT, t)

Say t is

```
new Pair(new A(), new B()).setfst(new B())
```

Then: t evaluates to `new Pair(new B(), new B())`

Substitute `new B()` for `newfst`

`new Pair(new A(), new B())` for `self` in

`body of setfst.`

Another example

Let t_1 be `new Pair(new A(), new B())`.

Let t_2 be `new Pair(t_1, new A()).fst`.

Let t_3 be `t_2.snd`

t_2 evaluates to t_1 . What does t_3 evaluate to?

What about types? t_1 : `Pair`, t_2 : `Object`. What is the type of t_3 ?

Another example

Let t_1 be `new Pair(new A(), new B())`.

Let t_2 be `new Pair(t1, new A()).fst`.

Let t_3 be `t2.snd`

t_2 evaluates to t_1 . What does t_3 evaluate to?

What about types? t_1 : `Pair`, t_2 : `Object`. What is the type of t_3 ?

To take care of typing: Let t_2 be `(Pair) new Pair(t1, new A()).fst`.

Evaluation: t_2 evaluates to `(Pair) t1`

`(Pair) t1` evaluates to `new Pair(new A(), new B())` after removal of the cast.

Now, t_3 evaluates to `new B()`.

Yet another example

```
class Point extends Object {
    int x; int y;
    Point(int x, int y){super(); self.x := x; self.y := y
    int getx() { return self.x; }
    int gety() { return self.y; }}

class ColorPoint extends Point {
    Color c;
    ColorPoint(int x, int y, color c){
        super(x, y); self.c := c;}
    Color getc() { return self.c; }}
```

BNF

```
CL ::= class C extends C{ $\bar{C}$   $\bar{f}$ ; K  $\bar{M}$ }
K  ::= C( $\bar{C}$   $\bar{f}$ ){super( $\bar{f}$ ); self. $\bar{f}$  :=  $\bar{f}$ ;}
M  ::= C m( $\bar{C}$   $\bar{x}$ ){return t;}
t  ::= x | t.f | t.m( $\bar{t}$ ) | new C( $\bar{t}$ ) | (C) t
v  ::= new C( $\bar{v}$ )
```

Notation: \bar{t} abbreviates the sequence t_1, t_2, \dots, t_n .

\bar{x} abbreviates the sequence x_1, x_2, \dots, x_n .

\bar{v} abbreviates the sequence v_1, v_2, \dots, v_n .

$\bar{C} \bar{f}$ abbreviates the sequence $C_1 f_1, C_2 f_2, \dots, C_n f_n$.

BNF (contd.)

$CL ::= \text{class } C \text{ extends } C\{\overline{C}; \overline{f}; \overline{K}; \overline{M}\}$

For example:

$\text{class } C \text{ extends } D\{C_1 f_1; C_2 f_2; \dots; C_n f_n \quad K M_1 M_2 M_3, \dots M_p\}$

Subtyping

Basic idea: $C \prec D$ – every value described by C is described by D , *i.e.*, any term of type C can safely be used in a context that expects a term of type D . Define \prec as:

$C \prec C$

$CT(C) = \text{class } C \text{ extends } D\{\dots\}$

$C \prec D$

$C \prec D \quad D \prec E$

$C \prec E$

Notes: **Object** does not occur in class table. **Object** has no fields, no methods. Any class $C \neq \text{Object}$ has a single super class.

Some notations

$C \vdash M$: Method declaration M is well-formed in class C .

$\Gamma \vdash t : C$. In type environment Γ , term t has type C .

$fields(C) = \overline{C.f}$. Fields of class C containing all fields declared in C and its superclasses.

$mtype(m, C)$. Type of method m declared or inherited in class C . (often written $\overline{B} \rightarrow B$, where \overline{B} is the type of the parameters, and B is the type of the result).

$C \text{ ok}$, $CT \text{ ok}$. Class C , class table CT resp. well-formed.

Typing: terms

$\Gamma \vdash x : C$ provided $(x : C) \in \Gamma$

$\Gamma \vdash t_0 : C_0 \quad \Gamma \vdash \bar{t} : \bar{C}$
 $mtype(m, C_0) = \bar{D} \rightarrow C$

$\Gamma \vdash t_0 : C_0 \quad fields(C_0) = \bar{C} \bar{f}$

$\bar{C} <: \bar{D}$

$\Gamma \vdash t_0.f_i : C_i$

$\Gamma \vdash t_0.m(\bar{t}) : C$

$fields(C) = \bar{D} \bar{f}$

$\Gamma \vdash \bar{t} : \bar{C} \quad \bar{C} <: \bar{D}$

$\Gamma \vdash t_0 : D$

$\Gamma \vdash_{new} C(\bar{t}) : C$

$\Gamma \vdash (C) t_0 : C$

Note: `self` $\in dom \Gamma$ always. Casts must be checked at run time.

mtime

`CT(C) = class C extends D{...; \bar{M} }`

`B m(\bar{B} \bar{x}){return t;} $\in \bar{M}$`

mtime(m, C) = $\bar{B} \rightarrow B$

`CT(C) = class C extends D{...; \bar{M} }`

`$m \notin \bar{M}$ mtime(m, D) = $\bar{B} \rightarrow B$`

mtime(m, C) = $\bar{B} \rightarrow B$

Typing: class declarations

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

fields(D) = $\bar{D} \ \bar{g} \quad C \vdash \bar{M}$

class C extends $D \{ \bar{C} \ \bar{f}; K \ \bar{M} \}$ ok

Typing: method declaration

`CT(C) = class C extends D{...}`

$\bar{x} : \bar{C}, \text{self} : C \vdash t : C_0$

`override(m, D, $\bar{C} \rightarrow C_0$)`

`C \vdash C0 m ($\bar{C} \bar{x}$) {return t;}`

mtype(m, D) = $\bar{D} \rightarrow D_0$ implies $\bar{C} = \bar{D}$ and $C_0 = D_0$

`override(m, D, $\bar{C} \rightarrow C_0$)`

Typing: class table and program

$$\frac{\forall C \in \text{dom}(\text{CT}), \text{CT}(C) \text{ ok}}{\text{CT ok}}$$

A program (CT, t) is well-formed iff CT is well-formed and

$$\vdash t : C.$$

One-step evaluation

$$\underline{\text{fields}(C) = \overline{C} \overline{f}}$$
$$\text{new } C(\overline{v}).f_i \rightarrow v_i$$
$$\underline{\text{mbody}(m, C) = (\overline{x}, e_0)}$$
$$\text{new } C(\overline{v}).m(\overline{d})$$
$$\rightarrow [\overline{x} \mapsto \overline{d}, \text{self} \mapsto \text{new } C(\overline{v})] e_0$$
$$\underline{C \prec: D \quad \text{type test at run time}}$$
$$(D) \text{ new } C(\overline{v}) \rightarrow \text{new } C(\overline{v})$$

Notation: $\text{mbody}(m, C)$. Body of method m declared or inherited in class C , written (\overline{x}, t) ,

where \bar{x} are the formal parameters, and t is the actual method body, *i.e.*, the “return”.

One-step evaluation, contd.

$$\frac{t_0 \rightarrow t'_0}{}$$

$$t_0.f \rightarrow t'_0.f$$

$$\bar{t} \rightarrow \bar{t}'$$

$$v_0.m(\bar{t}) \rightarrow v_0.m(\bar{t}')$$

$$t_0 \rightarrow t'_0$$

$$(C) t_0 \rightarrow (C) t'_0$$

$$t_0 \rightarrow t'_0$$

$$t_0.m(\bar{t}) \rightarrow t'_0.m(\bar{t})$$

$$\bar{t} \rightarrow \bar{t}'$$

$$\text{new } C(\bar{t}) \rightarrow \text{new } C(\bar{t}')$$

Handling parameter lists requires traversing the list till first non-value is found.

mbody

$CT(C) = \text{class } C \text{ extends } D\{\dots; \overline{M}\}$

$B \ m(\overline{B} \ \overline{x})\{\text{return } t;\} \in \overline{M}$

mbody(m, C) = (\overline{x}, t)

$CT(C) = \text{class } C \text{ extends } D\{\dots; \overline{M}\}$

$m \notin \overline{M} \quad \textit{mbody}(m, D) = (\overline{x}, t)$

mbody(m, C) = (\overline{x}, t)

Theorems

[Preservation]: Assume that CT is a well-formed class table.

If $\Gamma \vdash t : C$ and $t \rightarrow t'$ then $\Gamma \vdash t' : C'$ for some $C' \leq C$.

[Progress]: Assume that CT is a well-formed class table. If

$\vdash t : C$ then either:

1. t is a value or
2. t contains an expression of the form $(D) \text{new } C(\bar{v})$ where $C \not\leq D$ or
3. there exists t' such that $t \rightarrow t'$.