Software Engineering

Lecture 18: Featherweight Java

Peter Thiemann

University of Freiburg, Germany

15.07.2013

Contents

Type Safety of Java

- 1995 public presentation of Java
- Obtained importance very quickly
- Questions
  - Type safety?
  - Semantics of Java?
- 1997/98 resolved
  - Drossopoulou/Eisenbach
  - Flatt/Krishnamurthi/Felleisen
  - Igarashi/Pierce/Wadler (Featherweight Java, FJ)

Featherweight Java

- Construction of a formal model:
  consideration of completeness and compactness
- FJ: minimal model (compactness)
- complete definition: one page
- ambition:
  - the most important language features
  - short proof of type soundness
  - FJ $\subseteq$ Java

The Language FJ

- class definition
- object creation **new**
- method call (*dynamic dispatch*), recursion with **this**
- field access
- type cast
- method *override*
- subtypes

Omitted

- assignment
- interfaces
- *overloading*
- **super**-calls
- **null**-references
- primitive types
- abstract methods
- inner classes
- shadowing of fields of super classes
- access control (**private**, **public**, **protected**)
- *exceptions*
- concurrency
- reflection, generics, variable argument lists

Example Programs

```
class A extends Object { A() { super (); } }

class B extends Object { B() { super (); } }

class Pair extends Object {
  Object fst;
  Object snd;
  // Constructor
  Pair (Object fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
  // Method definition
  Pair setfst (Object newfst) {
    return new Pair (newfst, this.snd);
  }
}
```

Explanation

- Class definition: always define super class
- Constructors:
  - one per class, always defined
  - arguments correspond to fields
  - always the same form:
    super-call, then copy the arguments into the fields
- field accesses and method calls **always** with receiver object
- method body: always in the form **return**...

Examples for Evaluation

Method call

new Pair (**new** A(), **new** B()).setfst (**new** B())
// will be evaluated to
new Pair (**new** B(), **new** B())

Examples for Evaluation

Method call

```
new Pair (new A(), new B()).setfst (new B())
// will be evaluated to
new Pair (new B(), new B())
```

Type cast

```
((Pair) new Pair (new Pair (new A(), new B ()),
                  new A()).fst).snd
```

- Type cast (Pair) is needed, because **new** Pair (...).fst has the type Object.

Examples for Evaluation

Field access

```
new Pair (new A (), new B ()).snd
// will be evaluated to
new B()
```

Examples for Evaluation

Field access

new Pair (**new** A (), **new** B ()).snd
// will be evaluated to
new B()

Method call

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

yields a substitution

[**new** B()/newfst, **new** Pair (**new** A(), **new** B())/**this**]

Evaluate the method body **new** Pair (newfst, **this**.snd) under this substitution. The substitution yields

new Pair (**new** B(), **new** Pair (**new** A(), **new** B()).snd)

Examples of Evaluation

Type cast

```
(Pair)new Pair (new A (), new B ())
// evaluates to
new Pair (new A (), new B ())
```

- Run-time check if Pair is a subtype of Pair.

Examples of Evaluation

Type cast

```
(Pair)new Pair (new A (), new B ())
// evaluates to
new Pair (new A (), new B ())
```

- Run-time check if Pair is a subtype of Pair.

Call-by-value evaluation

```
((Pair) new Pair (new Pair (new A(), new B ()), new A()).fst).snd
// →
((Pair) new Pair (new A(), new B ())).snd
// →
new Pair (new A(), new B ()).snd
// →
new B()
```

Runtime Errors

Access to non existing field

```
new A().fst
```

No value, no evaluation rule matches

Runtime Errors

Access to non existing field

```
new A().fst
```

No value, no evaluation rule matches

Call of non-existing method

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

No value, no evaluation rule matches

Runtime Errors

Access to non existing field

```
new A().fst
```

No value, no evaluation rule matches

Call of non-existing method

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

No value, no evaluation rule matches

Failing type cast

```
(B)new A ()
```

- A is not subtype of B

⇒ no value, no evaluation rule matches

Guarantees of Java's Type System

If a Java program is type correct, then

- all field accesses refer to existing fields
- all method calls refer to existing methods,
- but failing type casts are possible.

Formal Definition

Syntax

$$
\begin{array}{lll}
CL & ::= & \text{class definition} \\
 & & \textbf{class}\ C\ \textbf{extends}\ D\ \{ C_1\ f_1; \ldots\ K\ M_1 \ldots \} \\
K & ::= & \text{constructor definition} \\
 & & C(C_1\ f_1, \ldots)\ \{\textbf{super}(g_1, \ldots); \textbf{this}.f_1 = f_1; \ldots\} \\
M & ::= & \text{method definition} \\
 & & C\ m(C_1\ x_1, \ldots)\ \{\textbf{return}\ t; \} \\
t & ::= & \text{expressions} \\
 & & x \qquad \text{variable} \\
 & & t.f \qquad \text{field access} \\
 & & t.m(t_1, \ldots) \qquad \text{method call} \\
 & & \textbf{new}\ C(t_1, \ldots) \qquad \text{object creation} \\
 & & (C)\ t \qquad \text{type cast} \\
v & ::= & \text{values} \\
 & & \textbf{new}\ C(v_1, \ldots) \qquad \text{object creation}
\end{array}
$$

Syntax—Conventions

- **this**
  - special variable, do not use it as field name or parameter
  - implicit bound in each method body
- sequences of field names, parameter names and method names include no repetition
- **class** C **extends** D $\{ C_1\ f_1; \ldots\ K\ M_1 \ldots \}$
  - defines class C as subclass of D
  - fields $f_1 \ldots$ with types $C_1 \ldots$
  - constructor K
  - methods $M_1 \ldots$
  - fields from D will be added to C, shadowing is not supported

Syntax—Conventions

- $C(D_1\ g_1, \dots, C_1\ f_1, \dots)\ \{\textbf{super}(g_1, \dots); \textbf{this}.f_1 = f_1; \dots\}$
  - define the constructor of class C
  - fully specified by the fields of C and the fields of the super classes.
  - number of parameters is equal to number of fields in C and all its super classes.
  - body start with **super**$(g_1, \dots)$, where $g_1, \dots$ corresponds to the fields of the super classes
- $D\ m(C_1\ x_1, \dots)\ \{\textbf{return}\ t;\}$
  - defines method m
  - result type D
  - parameter $x_1 \dots$ with types $C_1 \dots$
  - body is a **return** statement

Class Table

- The *class table* CT is a map from class names to class definitions
  - ⇒ each class has exactly one definition
  - the CT is global, it corresponds to the program
  - "arbitrary but fixed"
- Each class except Object has a superclass
  - Object is not part of CT
  - Object has no fields
  - Object has no methods ($\neq$ Java)
- The class table defines a subtype relation $C <: D$ over class names
  - the reflexive and transitive closure of subclass definitions.

Subtype Relation

$$\textsc{Refl}\quad \frac{}{C <: C}$$

$$\textsc{Trans}\quad \frac{C <: D \qquad D <: E}{C <: E}$$

$$\textsc{Ext}\quad \frac{CT(C) = \textbf{class}\ C\ \textbf{extends}\ D \ldots}{C <: D}$$

Subtype Relation

$$\text{REFL}\quad \frac{}{C <: C}$$

$$\text{TRANS}\quad \frac{C <: D \qquad D <: E}{C <: E}$$

$$\text{EXT}\quad \frac{CT(C) = \textbf{class}\ C\ \textbf{extends}\ D \ldots}{C <: D}$$

Java: **Assignment compatibility**

If $C <: D$, then

- a C-value can be assigned to a D-variable and
- a C-value can be passed as a D-parameter.

Intermezzo: Extension for Interfaces

Not part of FJ

$$\textsc{Impl}\ \frac{CT(C) = \textbf{class}\ C\ \textbf{implements}\ I \ldots}{C <: I}$$

$$\textsc{IExt}\ \frac{CT(I) = \textbf{interface}\ I\ \textbf{extends}\ J \ldots}{I <: J}$$

Consistency of CT

1. $CT(C) =$ **class** $C \ldots$ for all $C \in dom(CT)$
2. Object $\notin dom(CT)$
3. For each class name C mentioned in CT: $C \in dom(CT) \cup \{\text{Object}\}$
4. The relation <: is antisymmetric (no cycles)

Example: Classes Do Refer to Each Other

```
class Author extends Object {
  String name; Book bk;

  Author (String name, Book bk) {
    super();
    this.name = name;
    this.bk = bk;
  }
}
class Book extends Object {
  String title; Author ath;

  Book (String title, Author ath) {
    super();
    this.title = title;
    this.ath = ath;
  }
}
```

Auxiliary Definitions

Collect fields of classes

$$fields(\text{Object}) = \bullet$$

$$\frac{CT(C) = \textbf{class}\ C\ \textbf{extends}\ D\ \{C_1\ f_1; \ldots\ K\ M_1 \ldots\} \qquad fields(D) = D_1\ g_1, \ldots}{fields(C) = D_1\ g_1, \ldots, C_1\ f_1, \ldots}$$

- $\bullet$ — empty list
- *fields*(Author) = String name; Book bk;
- Usage: evaluation steps, typing rules

Auxiliary Definitions

Compute type of a method

$$\frac{CT(C) = \textbf{class } C \textbf{ extends } D \{ C_1\ f_1; \ldots\ K\ M_1 \ldots \} \qquad M_j = E\ m(E_1\ x_1, \ldots)\ \{\textbf{return } t;\}}{mtype(m, C) = (E_1, \ldots) \to E}$$

$$\frac{CT(C) = \textbf{class } C \textbf{ extends } D \{ C_1\ f_1; \ldots\ K\ M_1 \ldots \} \qquad (\forall j)\ M_j \neq F\ m(F_1\ x_1, \ldots)\ \{\textbf{return } t;\} \qquad mtype(m, D) = (E_1, \ldots) \to E}{mtype(m, C) = (E_1, \ldots) \to E}$$

- Usage: typing rules

Auxiliary Definitions

Determine body of a method

$$\frac{\begin{array}{c}CT(C) = \textbf{class } C \textbf{ extends } D\ \{C_1\ f_1; \ldots\ K\ M_1 \ldots\} \\ M_j = E\ m(E_1\ x_1, \ldots)\ \{\textbf{return } t;\}\end{array}}{mbody(m, C) = (x_1 \ldots, t)}$$

$$\frac{\begin{array}{c}CT(C) = \textbf{class } C \textbf{ extends } D\ \{C_1\ f_1; \ldots\ K\ M_1 \ldots\} \\ (\forall j)\ M_j \neq F\ m(F_1\ x_1, \ldots)\ \{\textbf{return } t;\} \qquad mbody(m, D) = (y_1 \ldots, u)\end{array}}{mbody(m, C) = (y_1 \ldots, u)}$$

- Usage: evaluation steps

Auxiliary Definitions

Correct overriding of a method

$$override(m, \text{Object}, (E_1 \dots) \to E)$$

$$\frac{\begin{array}{c} CT(C) = \textbf{class } C \textbf{ extends } D \; \{ C_1 \; f_1; \dots \; K \; M_1 \dots \} \\ M_j = E \; m(E_1 \; x_1, \dots) \; \{\textbf{return } t; \} \end{array}}{override(m, C, (E_1 \dots) \to E)}$$

$$\frac{\begin{array}{c} CT(C) = \textbf{class } C \textbf{ extends } D \; \{ C_1 \; f_1; \dots \; K \; M_1 \dots \} \\ (\forall j) \; M_j \neq F \; m(F_1 \; x_1, \dots) \; \{\textbf{return } t; \} \qquad override(m, D, (E_1, \dots) \to E) \end{array}}{override(m, C, (E_1, \dots) \to E)}$$

- Usage: typing rules

Example

```
class Recording extends Object {
    int high; int today; int low;
    Recording (int high, int today, int low) { ... }
    int dHigh() { return this.high; }
    int dLow() { return this.low }
    String unit() { return "not set"; }
    String asString() {
        return String.valueOf(high)
            .concat("–")
            .concat (String.valueOf(low))
            .concat (unit());
    }
}
class Temperature extends ARecording {
  Temperature (int high, int today, int low) { super(high, today, low); }
  String unit() { return "°C"; }
}
```

- $\mathit{fields}(\text{Object}) = \bullet$
- $\mathit{fields}(\text{Temperature}) = \mathit{fields}(\text{Recording}) = \text{int high; int today; int low;}$
- $\mathit{mtype}(\text{unit}, \text{Recording}) = () \to \text{String}$
- $\mathit{mtype}(\text{unit}, \text{Temperature}) = () \to \text{String}$
- $\mathit{mtype}(\text{dHigh}, \text{Recording}) = () \to \text{int}$
- $\mathit{mtype}(\text{dHigh}, \text{Temperature}) = () \to \text{int}$
- $\mathit{override}(\text{dHigh}, \text{Object}, () \to \text{int})$
- $\mathit{override}(\text{dHigh}, \text{Recording}, () \to \text{int})$
- $\mathit{override}(\text{dHigh}, \text{Temperature}, () \to \text{int})$
- $\mathit{mbody}(\text{unit}, \text{Recording}) = (\varepsilon, \text{"not set"})$
- $\mathit{mtype}(\text{unit}, \text{Temperature}) = (\varepsilon, \text{"}^\circ\text{C"})$

Operational Semantics (definition of the evaluation steps)

Direct Evaluation Steps

- Evaluation: relation $t \longrightarrow t'$ for one evaluation step

E-ProjNew
$$\frac{\mathit{fields}(C) = C_1\ f_1, \ldots}{(\textbf{new}\ C(v_1, \ldots)).f_i \longrightarrow v_i}$$

E-InvkNew
$$\frac{\mathit{mbody}(m, C) = (x_1 \ldots, t)}{\begin{array}{c}(\textbf{new}\ C(v_1, \ldots)).m(u_1, \ldots) \\ \longrightarrow t[\textbf{new}\ C(v_1, \ldots)/\text{this}, u_1, \ldots/x_1, \ldots]\end{array}}$$

E-CastNew
$$\frac{C <: D}{(D)(\textbf{new}\ C(v_1, \ldots)) \longrightarrow \textbf{new}\ C(v_1, \ldots)}$$

Evaluation Steps in Context

E-Field
$$\frac{t \longrightarrow t'}{t.f \longrightarrow t'.f}$$

E-Invk-Recv
$$\frac{t \longrightarrow t'}{t.m(t_1, \dots) \longrightarrow t'.m(t_1, \dots)}$$

E-Invk-Arg
$$\frac{t_i \longrightarrow t_i'}{v.m(v_1, \dots, t_i, \dots) \longrightarrow v.m(v_1, \dots, t_i', \dots)}$$

E-New-Arg
$$\frac{t_i \longrightarrow t_i'}{\textbf{new}\ C(v_1, \dots, t_i, \dots) \longrightarrow \textbf{new}\ C(v_1, \dots, t_i', \dots)}$$

E-Cast
$$\frac{t \longrightarrow t'}{(C)t \longrightarrow (C)t'}$$

Example: Evaluation Steps

((Pair) (**new** Pair (**new** Pair (**new** A(), **new** B()).setfst (**new** B()), **new** B()).fst)).fst

// → [E–Field], [E–Cast], [E–New–Arg], [E–InvkNew]

((Pair) (**new** Pair (**new** Pair (**new** B(), **new** B()), **new** B()).fst)).fst

// → [E–Field], [E–Cast], [E–ProjNew]

((Pair) (**new** Pair (**new** B(), **new** B()))).fst

// → [E–Field], [E–CastNew]

(**new** Pair (**new** B(), **new** B())).fst

// → [E–ProjNew]

new B()

Typing Rules

Typing Rules

Overview of typing judgments

- $C <: D$
  C is subtype of D
- $A \vdash t : C$
  Under type assumption A, the expression t has type C.
- $F\ m(C_1\ x_1, \dots)\ \{\mathbf{return}\ t;\}$ OK in C
  Method declaration is accepted in class C.
- $\mathbf{class}\ C\ \mathbf{extends}\ D\ \{C_1\ f_1; \dots\ K\ M_1 \dots\}$ OK
  Class declaration is accepted
- Type assumptions defined by

$$A ::= \emptyset \mid A, x : C$$

Accepted Class Declaration

$$\frac{\begin{array}{c} K = C(D_1\ g_1, \dots, C_1\ f_1, \dots)\ \{\textbf{super}(g_1, \dots); \textbf{this}.f_1 = f_1; \dots\} \\ \mathit{fields}(D) = D_1\ g_1 \dots \qquad (\forall j)\ M_j \text{ OK in } C \end{array}}{\textbf{class}\ C\ \textbf{extends}\ D\ \{C_1\ f_1; \dots\ K\ M_1 \dots\}}$$

Accepted Method Declaration

$$\frac{x_1 : C_1, \ldots, \text{this} : C \vdash t : E \qquad E <: F \qquad CT(C) = \textbf{class } C \textbf{ extends } D \ldots \qquad \mathit{override}(m, D, (C_1, \ldots) \to F)}{F\ m(C_1\ x_1, \ldots)\ \{\textbf{return } t;\}\ \text{OK in } C}$$

Expression Has Type

$$\textsc{T-Var}\quad \frac{x : C \in A}{A \vdash x : C}$$

$$\textsc{T-Field}\quad \frac{A \vdash t : C \qquad \mathit{fields}(C) = C_1\ f_1, \ldots}{A \vdash t.f_i : C_i}$$

$$\textsc{F-Invk}\quad \frac{A \vdash t : C \qquad (\forall i)\ A \vdash t_i : C_i \qquad (\forall i)\ C_i <: D_i \qquad \mathit{mtype}(m, C) = (D_1, \ldots) \to D}{A \vdash t.m(t_1, \ldots) : D}$$

$$\textsc{F-New}\quad \frac{(\forall i)\ A \vdash t_i : C_i \qquad (\forall i)\ C_i <: D_i \qquad \mathit{fields}(C) = D_1\ f_1, \ldots}{A \vdash \textbf{new}\ C(t_1, \ldots) : C}$$

Type Rules for Type Casts

T-UCast

$$\frac{A \vdash t : D \qquad D <: C}{A \vdash (C)t : C}$$

T-DCast

$$\frac{A \vdash t : D \qquad C <: D \qquad C \neq D}{A \vdash (C)t : C}$$

Type Safety for Featherweight Java

- "Preservation" and "Progress" yields type safety
- "Preservation":
  If $A \vdash t : C$ and $t \longrightarrow t'$, then $A \vdash t' : C'$ with $C' <: C$.
- "Progress": (short version)
  If $A \vdash t : C$, then $t \longrightarrow t'$, for some t', or $t \equiv v$ is a value, or t contains a subexpression e'

$$e' \equiv (C)(\textbf{new}\ D(v_1, \dots))$$

  with $D \not<: C$.

$\Rightarrow$
- All method calls and field accesses evaluate without errors.
- Type casts can fail.

Problems in the Preservation Proof

Type casts destroy preservation

- Consider the expression (A) ((Object)**new** B())
- It holds that $\emptyset \vdash$(A) ((Object)**new** B()): A
- It holds that (A) ((Object)**new** B()) $\longrightarrow$ (A) (**new** B())
- But (A) (**new** B()) has no type!

Problems in the Preservation Proof

Type casts destroy preservation

- Consider the expression (A) ((Object)**new** B())
- It holds that $\emptyset \vdash$(A) ((Object)**new** B()): A
- It holds that (A) ((Object)**new** B()) $\longrightarrow$ (A) (**new** B())
- But (A) (**new** B()) has no type!
- Workaround: add additional rule for this case "*stupid cast*" —subsequent evaluation step fails

$$\textsc{T-SCast}\quad \frac{A \vdash t : D \qquad C \not<: D \qquad D \not<: C}{A \vdash (C)t : C}$$

- We can prove preservation with this rule.

Statement of Type Safety

If $A \vdash t : C$, then one of the following cases applies:

1. t does not terminate
   i.e., there exists an infinite sequence of evaluation steps

$$t = t_0 \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \dots$$

2. t evaluates to a value v after a finite number of evaluation steps
   i.e., there exists a finite sequence of evaluation steps

$$t = t_0 \longrightarrow t_1 \longrightarrow \dots \longrightarrow t_n = v$$

3. t gets stuck at a failing cast
   i.e., there exists a finite sequence of evaluation steps

$$t = t_0 \longrightarrow t_1 \longrightarrow \dots \longrightarrow t_n$$

   where t_n contains a subterm $(C)(\textbf{new}\ D(v_1, \dots))$ such that $D \not<: C$.