Types of Java$_{\mathcal{O}}$

Types: Types A, B, C are:

- primitive types (`boolean`, `int`, `double`, ...)
- classes, interfaces
- `Null`, `void`
- If A is a type, $A \neq$ `Null` and $A \neq$ `void`, then $A[\,]$ is a type.

Reference types: classes, interfaces, `Null`, array types

Subtype relation for reference types: $A \preceq B$ is the least reflexive and transitive relation with:

- If $A \prec_{\mathrm{d}} B$, then $A \preceq B$.
- If A is a reference type, then `Null` $\preceq A$ and $A \preceq$ `Object`.
- $A[\,] \preceq$ `Cloneable` and $A[\,] \preceq$ `Serializable`.
- If $A \preceq B$ and A, B are reference types, then $A[\,] \preceq B[\,]$.

Properties of the subtype relation

- $A \preceq A$ [reflexive]
- $A \preceq B$ and $B \preceq C \implies A \preceq C$ [transitive]
- $A \preceq B$ and $B \preceq A \implies A = B$ [anti-symmetric]

For classes and interfaces A, B:

- $A \preceq B \iff A \preceq_{\mathrm{h}} B$ or $B =$ `Object`

Binary operators for references

Bop	Operand types	Result type	Operation
+	A or B is `String`	`String`	String concatenation
==	$A \preceq B$ or $B \preceq A$	`boolean`	equal (references)
!=	$A \preceq B$ or $B \preceq A$	`boolean`	not equal (references)

Syntax of Java$_{\mathcal{O}}$

$$
\begin{aligned}
\mathit{Exp} \;&:= \ldots \mid \texttt{null} \mid \texttt{this} \mid \mathit{Exp}.\mathit{Field} \mid \texttt{super}.\mathit{Field} \\
&\quad \mid \mathit{Exp}\ \texttt{instanceof}\ \mathit{Class} \mid (\mathit{Class})\mathit{Exp} \\
\mathit{Asgn} \;&:= \ldots \mid \mathit{Exp}.\mathit{Field} = \mathit{Exp} \mid \texttt{super}.\mathit{Field} = \mathit{Exp} \\
\mathit{Invk} \;&:= \ldots \mid \texttt{new}\ \mathit{Class}(\mathit{Exps}) \mid \mathit{Exp}.\mathit{Meth}(\mathit{Exps}) \mid \texttt{super}.\mathit{Meth}(\mathit{Exps})
\end{aligned}
$$

Constructor declarations in class A

$\langle$`public` | `protected` | `private`$\rangle$ $A(B_1\ loc_1, \ldots, B_n\ loc_n)\ cbody$

$cbody := block \mid \{$`this`$(exps);\ bstm\ \ldots\} \mid \{$`super`$(exps);\ bstm\ \ldots\}$

Default constructor: $A()\ \{$`super`$();\ \}$

Example:

```
class A {
  private int x;
  private int y = 17;
  static int z = 3;

  A(int x) {
    this.x = x;
  }
}
```

Compiler:

```
A(int x) {
  super();
  y = 17;
  this.x = x;
}

static { z = 3; }
```

Instance field access expressions in class A

$$\left.\begin{array}{l} exp.field \\ \texttt{super}.field \\ field \end{array}\right\} \Longrightarrow exp.C/field \qquad \text{[Compiler]}$$

Case 1: If $B = \mathcal{T}(exp)$ and C is unique with

- $field$ is declared in C
- $C/field$ is visible in B and accessible from A with respect to B

Case 2: If $A \prec_{\mathrm{d}} B$ and C is unique with

- $field$ is declared in C
- $C/field$ is visible in B and accessible from A

Case 3: Similar to $\texttt{this}.field$, if $field$ is not in the scope of a local variable declaration of $field$.

Syntaxerror: $field$ is not `static` in C. (?)

Instance method invocation expressions in class A

$$\left.\begin{array}{l} {}^{\alpha}({}^{\beta}exp.meth^{\gamma}(exps)) \\ {}^{\alpha}\texttt{super}.meth^{\gamma}(exps) \\ {}^{\alpha}meth^{\gamma}(exps) \end{array}\right\} \Longrightarrow {}^{\alpha}({}^{\beta}exp.D/m^{\gamma}(exps)) \qquad \text{[Compiler]}$$

Let $msig = meth(\mathcal{T}(\gamma))$.

Case 1: Let $C = \mathcal{T}(\beta)$. Then $D/m \in app(\alpha)$ iff

1. $C/msig$ is more specific than D/m
2. D/m is visible in C and accessible from A with respect to C
3. if D is an interface, then C does not implement m

Case 2: Let $A \prec_{\mathrm{d}} C$. $D/m \in app(\alpha)$ iff

1. $C/msig$ is more specific than D/m
2. D/m is visible in C and accessible from A
3. if D is an interface, then C does not implement m

Case 3: Similar to ${}^{\alpha}({}^{\beta}\texttt{this}.meth^{\gamma}(exps))$.

Method invocation expressions (coninued)

Constraint: D/m must be the most specific method in $app(\alpha)$.

Invocation mode: *CallKind* $:= \mathit{Virtual} \mid \mathit{Special} \mid \mathit{Super}$

Case 1:

$$callKind(\alpha) := \begin{cases} \mathit{Special}, & \text{if } m \text{ is } \texttt{private} \text{ in } D; \\ \mathit{Virtual}, & \text{otherwise} \end{cases}$$

Case 2: ${}^{\beta}exp$ is ${}^{\beta}$`this` and $callKind(\alpha) := \mathit{Super}$

Case 3: Similar to Case 1, if m is not `static` in D.

$\mathit{Virtual}$	$\Longrightarrow$	dynamic method lookup (late binding)
$\mathit{Special}$	$\Longrightarrow$	direct call (early binding)
Super	$\Longrightarrow$	dynamic method lookup starting in C

Example: Applicable methods

```
interface I { void m(int i); }
class A {
  public void m(int i) { }
}
class B extends A implements I {
  void test(int i) {
    αm(i);
  }
}
```

Remark: The method $\mathtt{I}/\mathtt{m}(\mathtt{int})$ is not applicable at α.

Remark: $\mathtt{A}/\mathtt{m}(\mathtt{int})$ is not more specific than $\mathtt{I}/\mathtt{m}(\mathtt{int})$.

Example: Ambiguity?

```
interface I {
  void m(J x);
}
interface J extends I {
  void m(I x);
}
abstract class A implements J {
  void test(J x) {
   this.m(x);
   ((A)x).m(x);
   x.m(x);
  }
}
```

```
JDK 1.3
javac A.java
```

```
abstract class I {
  abstract void m(J x);
}
abstract class J extends I {
  abstract void m(I x);
}
abstract class A extends J {
  void test(J x) {
    this.m(x);
    ((A)x).m(x);
    x.m(x);
  }
}
```

```
JDK 1.3
javac A.java
reference to m is ambiguous: this.m(x);
reference to m is ambiguous: ((A)x).m(x);
reference to m is ambiguous: x.m(x);
3 errors
```

Instance creation expressions

$\texttt{new}\ C(exps) \implies (\texttt{new}\ C).C/msig(exps)$

where $msig = \texttt{<init>}(A_1, \ldots, A_n)$ and $callKind = Special$

Type constraints for Java$_{\mathcal{O}}$

${}^{\alpha}\texttt{null}$	$\mathcal{T}(\alpha) = \texttt{Null}$
${}^{\alpha}\texttt{this}$	$\mathcal{T}(\alpha) = A$, if the position α is in class A.
${}^{\alpha}({}^{\beta}exp\ \texttt{instanceof}\ A)$	$\mathcal{T}(\alpha) = \texttt{boolean}$. A is a reference type. $(*)$ It must be possible that there is a class or array type C with $C \preceq A$ and $C \preceq \mathcal{T}(\beta)$.
${}^{\alpha}((A)\,{}^{\beta}exp)$	$\mathcal{T}(\alpha) = A$. A is a reference type. $(*)$
${}^{\alpha}(exp.C/\mathit{field})$	$\mathcal{T}(\alpha)$ is the declared type of field in class C.
${}^{\alpha}(exp_1.C/\mathit{field} = {}^{\gamma}exp_2)$	$\mathcal{T}(\alpha)$ is the declared type of field in C, field is not $\texttt{final}$ in C, $\mathcal{T}(\gamma) \preceq \mathcal{T}(\alpha)$.
${}^{\alpha}\texttt{new}\ C.C/msig(exps)$	$\mathcal{T}(\alpha) = C$. C a class. C not $\texttt{abstract}$.
${}^{\alpha}(exp.C/msig(exps))$	$\mathcal{T}(\alpha)$ is the declared return type of method $msig$ in class or interface C.

Vocabulary of the ASM for Java$_{\mathcal{O}}$

Static functions:

$$
\begin{aligned}
instanceFields &: \mathsf{Class} \to Powerset(\mathsf{Class/Field}) \\
defaultVal &: \mathsf{Type} \to \mathsf{Val} \\
type &: \mathsf{Class/Field} \to \mathsf{Type} \\
lookup &: (\mathsf{Class}, \mathsf{Class/MSig}) \to \mathsf{Class}
\end{aligned}
$$

Example (Instance fields):

```
class A {
  private int x;
  public int y;
  public static int z;
}
class B extends A {
  private int x;
}
```

$instanceFields(\mathtt{B}) = [\mathtt{A/x}, \mathtt{A/y}, \mathtt{B/x}]$.

Dynamic method lookup

$lookup(A, B/msig) =$

if A contains a non `abstract` declaration of $msig$ **and**
(B is an interface **or** $A/msig$ overrides $B/msig$)
then
A
else if $A =$ `Object` **then**
$undef$
else
$lookup(super(A), B/msig)$

Vocabulary of the ASM for Java$_{\mathcal{O}}$ (continued)

Universes:

Ref . . . . references, pointers

Val . . . . values ($Val = \ldots \mid Ref \mid null$)

Heap . . . . objects (instances of classes) and arrays

$$Heap = Object(Class, Map(Class/Field, Val)) \mid Array(Type, [Val])$$

Dynamic function:

$heap\colon Ref \to Heap$

Run-time type:

$$\begin{aligned} classOf(ref) = \textbf{case } & heap(ref) \textbf{ of} \\ & Array(t, elems) \to t[\,] \\ & Object(c, fields) \to c \end{aligned}$$

Initial state of Java$_{\mathcal{O}}$: $heap = \emptyset$

Transition rules for Java$_{\mathcal{O}}$

$$execJava_O = $$
$$\quad execJavaExp_O$$

$$getField(ref, f) = fields(f)$$
$$\quad \textbf{where } Object(c, fields) = heap(ref)$$

$$setField(ref, f, val) =$$
$$\quad heap(ref) := Object(c, fields \oplus \{(f, val)\})$$
$$\quad \textbf{where } Object(c, fields) = heap(ref)$$

$$exitMethod(result) =$$
$$\quad \ldots$$
$$\quad \textbf{elseif } methNm(meth) = \texttt{"<init>"} \wedge result = Norm \textbf{ then}$$
$$\quad\quad restbody := oldPgm[locals(\texttt{"this"})/oldPos]$$
$$\quad \ldots$$

Transition rules for Java$_{\mathcal{O}}$ (continued)

$$
\begin{array}{l}
execJavaExp_{O} = \textbf{case } context(pos) \textbf{ of} \\
\quad \texttt{this} \rightarrow yield(locals(\texttt{"this"})) \\
\quad \texttt{new } c \rightarrow \textbf{if } initialized(c) \textbf{ then create } ref \\
\qquad\qquad heap(ref) := Object(c, \{(f, defaultVal(type(f))) \\
\qquad\qquad\qquad\qquad\qquad \mid f \in instanceFields(c)\}) \\
\qquad\qquad yield(ref) \\
\qquad\quad \textbf{else } initialize(c) \\
\quad {}^{\alpha}exp.c/f \rightarrow pos := \alpha \\
\quad {}^{\blacktriangleright}ref.c/f \rightarrow \textbf{if } ref \neq null \textbf{ then } yieldUp(getField(ref, c/f)) \\
\quad {}^{\alpha}exp_1.c/f = {}^{\beta}exp_2 \rightarrow pos := \alpha \\
\quad {}^{\blacktriangleright}ref.c/f = {}^{\beta}exp \quad \rightarrow pos := \beta \\
\quad {}^{\alpha}ref.c/f = {}^{\blacktriangleright}val \quad \rightarrow \textbf{if } ref \neq null \textbf{ then} \\
\qquad setField(ref, c/f, val) \\
\qquad yieldUp(val)
\end{array}
$$

Transition rules for Java$_{\mathcal{O}}$ (continued)

$$^{\alpha}exp\ \texttt{instanceof}\ c \rightarrow pos := \alpha$$
$$^{\blacktriangleright}ref\ \texttt{instanceof}\ c \rightarrow yieldUp(ref \neq null \wedge classOf(ref) \preceq c)$$

$$(c)^{\alpha}exp \rightarrow pos := \alpha$$
$$(c)^{\blacktriangleright}ref \rightarrow \textbf{if}\ ref = null \vee classOf(ref) \preceq c\ \textbf{then}\ yieldUp(ref)$$

$$
\begin{array}{l}
^{\alpha}exp.c/m^{\beta}(exps) \rightarrow pos := \alpha \\
^{\blacktriangleright}ref.c/m^{\beta}(exps) \rightarrow pos := \beta \\
^{\alpha}ref.c/m^{\blacktriangleright}(vals) \rightarrow \textbf{if}\ ref \neq null\ \textbf{then} \\
\quad \textbf{let}\ c' = \textbf{case}\ callKind(up(pos))\ \textbf{of} \\
\qquad\qquad\qquad \begin{array}{ll} Virtual & \rightarrow lookup(classOf(ref), c/m) \\ Super & \rightarrow lookup(super(classNm(meth)), c/m) \\ Special & \rightarrow c \end{array} \\
\quad invokeMethod(up(pos), c'/m, [ref] \cdot vals)
\end{array}
$$