

cd ::= class C extends D { \overline{fd} ; kd \overline{md} }	class
fd ::= q C f	field
kd ::= q C (t C g, t C f) { super(g); this.f = f; }	constructor
md ::= t C m (t C this, t C x) { t C y s; return z }	instance method
e ::= x x.f	expression
s ::= x = e x.f = y x = y.m(z) super(g) this(g) x = new C() s;s	statement
t ::= k q	qualifier type
k ::= initialized underinitialized unknowninitialized fbcbottom	initialization qualifier
q ::= readonly mutable polymutable substitutablepolymutable receiverdependantmutable immutable bottom	immutability qualifier

Qualifier Hierarchy

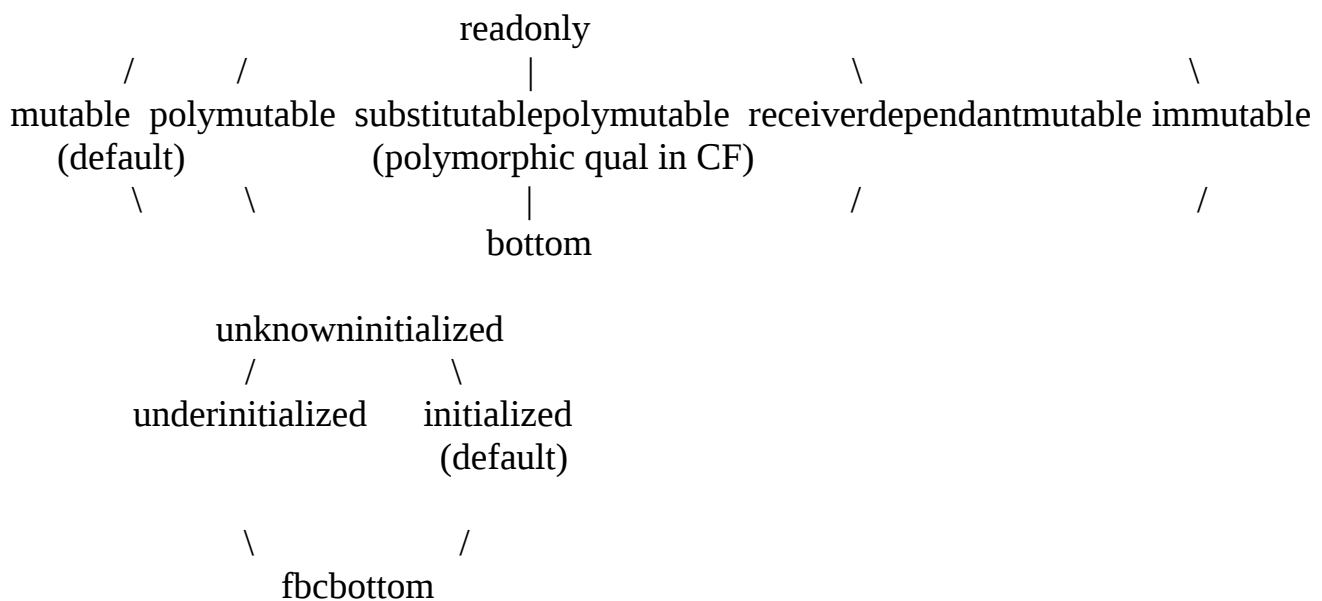


Figure 1 Combination of qualifiers. Two qualifier hierarchies are orthogonal. If an object is under initialization, its immutability guarantee is not satisfied. So even immutable and receiverdependantmutable objects can also be modified when under initialization.

Subtype relations

$$k_1 \ q_1 <: k_2 \ q_2 \iff k_1 <: k_2 \ \wedge \ q_1 <: q_2$$

Helper Functions

$q \ C \ f$

$fType(f) = q$

Note:

- 1) No initialization modifier on field declarations. In actual implementation, to have circular initialization, *@NotOnlyInitialized* can be used on field declaration. However, it doesn't belong to initialization qualifier hierarchy.
- 2) The field is unique within the whole type hierarchy

$fields(C)$ returns all fields directly declared in C .

$cBody(kd)$ returns constructor body of kd .

$mBody(md)$ returns method body of md .

Viewpoint Adaptation Rules

$_ \triangleright mutable = mutable$
 $_ \triangleright readonly = readonly$
 $_ \triangleright immutable = immutable$
 $_ \triangleright bottom = bottom$
 $_ \triangleright polymutable = substitutablepolymutable$
 $q \triangleright receiverdependantmutable = q$

Note: substitutablepolymutable only exists shortly after viewpoint adaptation is done, but will must be substituted by another qualifier immediately by QualifierPolymorphism. So, substitutablepolymutable should not appear on left or right side of viewpoint adaptation triangle.

Special Rules

- Forbid polymutable fields; readonly or polymutable constructor return type and readonly instantiation of objects
- In constructor, $q_{this} = q_{ret}$
- Forbid initialization modifier on fields, constructor return type and new statement
- Forbid bottom except on (implicit/explicit) lower bounds and null literal.
- Forbid explicit use of substitutablepolymutable everywhere.

TODO: Should we allow polymutable constructor return type?

Typing Rules

$$\frac{x \in \Gamma}{\Gamma \vdash x : \Gamma(x)} \text{ (T-VAR)}$$

$$\frac{\begin{array}{l} \Gamma(x) = k_x q_x \quad \text{fType}(f) = q_f \quad q = q_x \triangleright q_f \\ k = \begin{cases} \text{initialized} & \text{if } k_x = \text{initialized} \\ \text{unknowninitialized} & \text{otherwise} \end{cases} \end{array}}{\Gamma \vdash x.f : k q} \text{ (T-FLD)}$$

Figure 2 Expression typing

$$\frac{\Gamma \vdash e = t_e \quad t_e <: \Gamma(x)}{\Gamma \vdash x = e} \text{ (T-VARASS)}$$

$$\frac{\begin{array}{l} \Gamma(x) = k_x q_x \quad \Gamma(y) = k_y q_y \quad \text{typeof}(f) = q_f \\ q_x = \text{mutable} \vee \\ q_f = \text{mutable} \vee \\ q_f = \text{readonly} \vee \\ (k_x = \text{underinitialized} \wedge q_x = \text{immutable}) \vee \\ (k_x = \text{underinitialized} \wedge q_x = \text{polyimmutable}) \\ q_y <: q_x \triangleright q_f \\ k_x = \text{underinitialized} \vee k_y = \text{initialized} \end{array}}{\Gamma \vdash x.f = y} \text{ (T-FLDASS)}$$

$$\Gamma(x) = k_x \ q_x \quad \Gamma(y) = k_y \ q_y \quad \Gamma(\bar{z}) = \bar{k}_z \ \bar{q}_z \quad \text{typeof}(m) = k_{\text{this}} \ q_{\text{this}}, \ \bar{k}_p \ \bar{q}_p \rightarrow k_{\text{ret}} \ q_{\text{ret}}$$

$$k_y <: k_{\text{this}} \quad \bar{k}_z <: \bar{k}_p \quad k_{\text{ret}} <: k_x$$

$$q_{\text{this-vp}} = q_y \triangleright q_{\text{this}} \quad \bar{q}_{p\text{-vp}} = q_y \triangleright \bar{q}_p \quad q_{\text{ret-vp}} = q_y \triangleright q_{\text{ret}}$$

$$q_{\text{this-vp}} = \text{substitutablepolymutable} \vee \bar{q}_{p\text{-vp}} = \text{substitutablepolymutable} \vee q_{\text{ret-vp}} = \text{substitutablepolymutable} \Rightarrow s \text{ exists}$$

$$q_y <: \begin{cases} q_{\text{this-vp}} & \text{if } q_{\text{this-vp}} \neq \text{substitutablepolymutable} \\ s & \text{else} \end{cases}$$

$$\bar{q}_z <: \begin{cases} \bar{q}_{p\text{-vp}} & \text{if } \bar{q}_{p\text{-vp}} \neq \text{substitutablepolymutable} \\ s & \text{else} \end{cases}$$

$$q_x >: \begin{cases} q_{\text{ret-vp}} & \text{if } q_{\text{ret-vp}} \neq \text{substitutablepolymutable} \\ s & \text{else} \end{cases}$$

$$\Gamma \vdash x = y.[s]m(\bar{z}) \quad (\text{T-CALL})$$

Note: inference of s is another subproblem. It is disussed in the last two pages.

$$kd \text{ in } C \quad C <: D \quad \text{typeof}(D) = \bar{k}_{p\text{-D}} \ \bar{q}_{p\text{-D}} \rightarrow q_{\text{ret-D}} \quad \text{typeof}(kd) = \bar{_} \bar{_} \rightarrow q_{\text{ret-C}}$$

$$q_{\text{ret-C}} = \begin{cases} \text{immutable} & \text{if } q_{\text{ret-D}} = \text{immutable} \\ \text{mutable} & \text{if } q_{\text{ret-D}} = \text{mutable} \end{cases}$$

$$\Gamma(z) = k_z \ q_z \quad \bar{k}_z <: \bar{k}_{p\text{-D}} \quad \bar{q}_z <: q_{\text{ret-C}} \triangleright \bar{q}_{p\text{-D}}$$

$$\Gamma \vdash \text{super}(\bar{z}) \text{ in } kd \quad (\text{T-SUPER})$$

* Previously, when $q_{\text{ret-D}} = \text{mutable}$, $q_{\text{ret-C}}$ can still be immutable. Because at that time, immutable constructors only have immutable or polyimmutable(does not exist anymore), thus any mutable objet created locally cannot escape and be captured by outside objects; Neither outside mutable objects will be captured by the

receiverdependantmutable field when invoking mutable super constructor in immutable constructor. But now, immutable and receiverdependantmutable constructors don't have such restrictions(mutable parameters are allowed in both cases) any more, so outside mutable objects can be captured by receiverdependantmutable field. If we allow calling mutable super() in immutable subclass constructor, when we use $\text{this}_{\text{sub}}.\text{rdmf}$ to access the field, the result is not guarantee to be immutable(may be the mutable object assigned in super mutable constructor). Therefore, we don't allow this kinds of flexibility and require subclass and superclass constructors should have the exact same qualifier if $q_{\text{ret-D}} \neq \text{receiverdependantmutable}$

(T-THIS) (omitted)

* *Note:* In real Java code, one class can have multiple overloaded constructors. One constructor can invoke the other by “this(..., ...)”. The type rule T-THIS is very much the same as T-SUPER except that the constructor invoked by “this(..., ...)” comes from the same class.

$$\begin{array}{lll} \Gamma(\underline{x}) = k_x \ q_x & \Gamma(\bar{y}) = \bar{k}_y \ \bar{q}_y & \text{typeof}(C) = \bar{k}_p \ \bar{q}_p \rightarrow q_{\text{ret}} \\ q_y <: q \triangleright q_p & q <: q \triangleright q_{\text{ret}} & q \neq \text{readonly} \end{array}$$

$$\bar{k}_y <: \bar{k}_p$$

$$q <: q_x \quad k <: k_x \quad k = \begin{cases} \text{initialized} & \text{if } \bar{k}_p = \text{initialized} \\ \text{underinitialized} & \text{otherwise} \end{cases}$$

$$\Gamma \vdash x = \text{new } q \ C(\bar{y})$$

(T-NEW)

$$\Gamma \vdash s_1 \quad \Gamma \vdash s_2$$

$$\Gamma \vdash s_1; s_2$$

(T-SEQ)

Figure 3 Statement typing

Well-formdness Rules

$fType(fd) \neq \text{polymutable} \quad C <: D \quad fd \notin \text{fields}(D)$

$\vdash_C fd \text{ is OK}$

(WF-FLD)

$\vdash_{\text{object}} kd \text{ is OK}$

(WF-CONS-OBJECT)

$cBody(kd) = \text{super}(g); \text{this.f} = f \quad \text{typeof}(kd) = \bar{k}_p \bar{q}_p \rightarrow q_{\text{ret}}$
 $q_{\text{ret}} \neq \text{readonly} \wedge q_{\text{ret}} \neq \text{polymutable}$
 $\Gamma = (\text{this} : \text{underinitialized } q_{\text{ret}}, \bar{p} : \bar{k}_p \bar{q}_p, \bar{y} : \bar{k}_{\text{local}} \bar{q}_{\text{local}})$
 $\Gamma \vdash \text{super}(\bar{y}) \text{ in } kd \quad \Gamma \vdash \text{this.f} = f$

$\vdash_C kd \text{ is OK}$

(WF-CONS)

Note: $\vdash_C kd$ reads “constructor kd in class C is well-formed”.

$mBody(md) = \bar{s}; \text{return } z \quad \text{typeof}(md) = k_{\text{this}} q_{\text{this}}, \bar{k}_p \bar{q}_p \rightarrow t_{\text{ret}}$
 $\Gamma = (\text{this} : k_{\text{this}} q_{\text{this}}, \bar{p} : \bar{k}_p \bar{q}_p, \bar{y} : \bar{k}_{\text{local}} \bar{q}_{\text{local}}) \quad \Gamma \vdash \bar{s} \quad \Gamma(z) <: t_{\text{ret}}$
 Standard method overriding rule

$\vdash_C md \text{ is OK}$

(WF-METH)

$$\begin{array}{c}
\vdash_c \overline{fd} \text{ is OK} \quad \vdash_c \overline{kd} \text{ is OK} \quad \vdash_c \overline{md} \text{ is OK} \\
\hline
\vdash C \text{ is OK}
\end{array}
\quad (\text{WF-CLASS})$$

Figure 4 Well-formdness typing

Extension to real Java

In real Java, there are static fields, static methods, initialization blocks.

Helper Method

usedQualifiers(\bar{s}) returns all immutability qualifiers used in \bar{s} recursively

$cd ::= \text{class } C \text{ extends } D \{ \overline{sfd} \overline{fd}; \overline{sib} \overline{ib} \overline{kd} \overline{smd} \overline{md} \}$	class
$sfd ::= \text{static } q \ C \ sf$	static field
$smd ::= \text{static } \underline{t} \ C \ sm \ (\ \underline{t} \ C \ x \) \ \{ \ \overline{t} \ C \ y \ \bar{s}; \ \text{return } z; \}$	static method
$sib ::= \text{static}\{\bar{s};\}$	static initialization block
$ib ::= \{\bar{s};\}$	initialization block

$$\begin{array}{c}
fType(sfd) \neq \text{polymutable} \wedge fType(sfd) \neq \text{receiverdependantmutable} \\
\hline
\vdash sfd \text{ is OK}
\end{array}
\quad (\text{WF-STATIC-FLD})$$

$$\begin{array}{c}
mBody(smd) = \bar{s}; \text{return } z \quad \text{typeof}(smd) = \bar{k}_p \ \bar{q}_p \rightarrow t_{ret} \\
\Gamma = (\underline{p} : \bar{k}_p \ \bar{q}_p, \ \underline{y} : \bar{k}_{local} \ \bar{q}_{local}) \quad \Gamma \vdash \bar{s} \quad \Gamma(z) <: t_{ret} \\
\bar{q}_p \neq \text{receiverdependantmutable} \wedge \bar{q}_{ret} \neq \text{receiverdependantmutable} \\
\text{receiverdependantmutable} \notin \text{usedQualifiers}(\bar{s}; \text{return } z) \\
\hline
\vdash smd \text{ is OK}
\end{array}$$

(WF-STATIC-METH)

$\Gamma \vdash \bar{s} \quad \text{receiverdependantmutable} \notin \text{usedQualifiers}(\bar{s})$	
$\vdash \text{ sib is OK}$	(WF-STATIC-BLK)
$\Gamma \vdash \bar{s}$	
$\vdash_c \text{ ib is OK}$	(WF-BLK)
$\vdash \overline{\text{sfd}} \text{ is OK} \quad \vdash_c \overline{\text{fd}} \text{ is OK} \quad \vdash_c \text{ kd is OK} \quad \vdash \overline{\text{smd}} \text{ is OK} \quad \vdash_c \overline{\text{md}} \text{ is OK}$ $\vdash \overline{\text{sib}} \text{ is OK} \quad \vdash_c \overline{\text{ib}} \text{ is OK}$	
$\vdash C \text{ is OK}$	(WF-CLASS)

Inference of immutability qualifier for polymutable methods

After viewpoint adapting $m()$ at the invocation site, if $q_{\text{this-vp}}, \bar{q}_{\text{p-vp}}, q_{\text{ret-vp}}$ are NOT substitutablepolymutable, standard subtyping rules apply:

$$q_y <: q_{\text{this-vp}} \quad \bar{q}_z <: \bar{q}_{\text{p-vp}} \quad q_{\text{ret-vp}} <: q_x$$

But if any of them is substitutablepolymutable, we use a variable s to replace corresponding $q_{\text{this-vp}}, \bar{q}_{\text{p-vp}}, q_{\text{ret-vp}}$ and add it/them to constraints set. After collecting all the constraints, we try to find a solution s that satisfies all the subtype constraints. If there is such a solution, then method invocation typechecks; Otherwise, it doesn't typecheck.

For example, assuming we have a method after viewpoint adaptation with signature:
substitutablepolymutable Object $m(\text{ substitutablepolymutable A this, substitutablepolymutable Object p})$;

If we invoke it as:

immutable A a ;

readonly Object $ro = a.m(\text{new immutable Object}());$

Constraints are collected in this way:

$$\text{immutable} <: s \quad \text{immutable} <: s \quad s <: \text{readonly}$$

We'll have a solution:

$s = \text{immutable(or readonly)}$, so this method invocation typechecks.

But if we invoke the method as:


```
mutable Object ro = a.m(new immutable Object());
```

Then we have constraints:

```
immutable <: s      immutable <: s      s <: mutable
```

There is NO solution for s, so the type system rejects this method invocation.