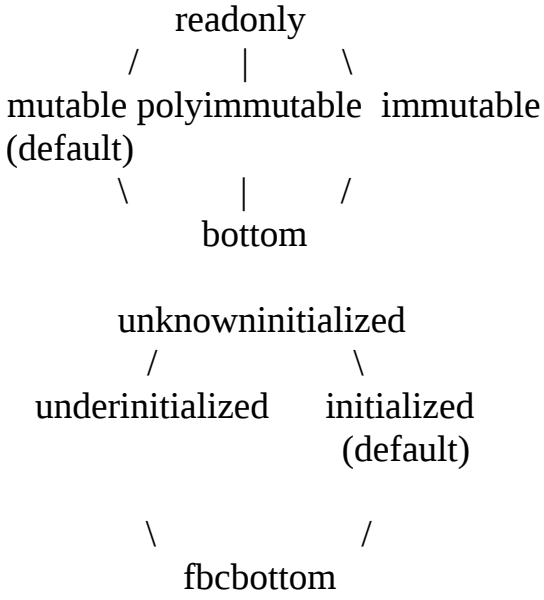
cd ::= class C extends D { $\overline{fd}$; kd $\overline{md}$ }	class
fd ::= q C f	field
kd ::= q C (t C g, t C f) { super(g); this.f = f; }	constructor
md ::= t C m (t C this, t C x) { $\overline{t\ C\ y\ s}$; return z }	instance method
e ::= x | x.f	expression
s ::= x = e | x.f = y | x = y.m(z) |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

Each class has only one constructor. But it doesn't affect the generality.

Type Hierarchy



	Mutable	Immutable	PolyImmutable	Readonly
UnderInitialized	√	√	√	X
Initialized	√	X	X	X
UnknownInitializaed	√	X	X	X

√ means allowing assigning fields

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 polyimmutable objects can also be modifed when under initialization. We don't have readonly objects, so there is no need to initialize readonly objects. Therefore, readonly doesn't have such exception when under initialization.

Subtype relations:

$k_1\ q_1 <: k_2\ q_2 <=> k_1 <: k_2 \wedge q_1 <: q_2$

Helper Functions

$$\frac{q\ C\ f}{fType(f) = q}$$

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

_ ▷ mutable = mutable
_ ▷ readonly = readonly
_ ▷ immutable = immutable
_ ▷ bottom = bottom
q ▷ polyimmutable = q

Special Rules

- Forbid mutable fields, readonly 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.

Typing Rules

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

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

Figure 2 Expression typing

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

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

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

In definition of m(), if any of $q_{this}, \bar{q}_p, q_{ret}$ is NOT @PolyImmutable, standard subtyping rules apply:

$$q_y <: q_{this} \qquad \bar{q}_z <: \bar{q}_p \qquad q_{ret} <: q_x$$

If any of them is @PolyImmutable, we use @PolyImmutable to replace corresponding $q_{this}, \bar{q}_p, q_{ret}$ and add it/them to constraints set. If there is solution, then method invocation typechecks; Otherwise, it doesn't typecheck. For example, assuming we have a method with signature:

@PolyImmutable Object m(@PolyImmutable A this, @PolyImmutable Object p);
Constraints are collected in this way:
q_y <: @PolyImmutable $\bar{q}_z$ <: @PolyImmutable @PolyImmutable <: q_x

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

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

$q_{ret\text{-}C}$ = { _ *if* $q_{ret\text{-}D}$ = *polyimmutable* ∨ *mutable*; immutable *otherwise*

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

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

* *Note:* In real Java code, one class can have multiple overloaded consturctors. 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.

$\Gamma(x) = k_x\ q_x$ $\Gamma(\bar{y}) = \bar{k}_y\ \bar{q}_y$ typeof(C) = $\bar{k}_p\ \bar{q}_p \rightarrow q_{ret}$
$\bar{q}_y$ <: $q \triangleright q_p$ q <: $q \triangleright q_{ret}$ q = mutable ∨ q = immutable ∨ q = polyimmutable

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

q <: q_x k <: k_x k = { initialized *if* $\bar{k}_p$ = *initialized*; underinitialized *otherwise*

(T-NEW)

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

$$\frac{\Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash s_1;s_2} \qquad \text{(T-SEQ)}$$

Figure 3 Statement typing

Well-formdness Rules

$$\frac{fType(f) \neq mutable \qquad C <: D \qquad f \notin fields(D)}{\vdash_C f \text{ is OK}} \text{(WF-FLD)}$$

$$\frac{}{\vdash_{object} kd \text{ is OK}} \text{(WF-CONS-OBJECT)}$$

$$\frac{cBody(kd) = super(g);s \qquad typeof(kd) = \overline{k_p}\ \overline{q_p} \rightarrow q_{ret} \qquad q_{ret} \neq readonly \qquad \Gamma = (this : underinitialized\ q_{ret},\ \overline{p : k_p\ q_p},\ \overline{y : k_{local}\ q_{local}}) \qquad \Gamma \vdash super(\overline{y}) \text{ in } kd \qquad \Gamma \vdash s \qquad \overline{q_p} = \begin{cases} \text{polyimmutable or immutable} & \text{if } q_{ret} = polyimmutable \text{ or } q_{ret} = immutable \\ \_ & otherwise \end{cases}}{\vdash_C kd \text{ is OK}} \text{(WF-CONS)}$$

Note: $\vdash_{C\ kd}$ reads "constructor kd in class C is well-formed".

Only allowing polyimmutable and immutable constructor parameter types in polyimmutable and immutable constructor allows readonly field to be safe, i.e., no aliased mutable objects will be captured by readonly fields of an immutable object and break the immutability contract.

$$\frac{mBody(md) = s;return\ z \qquad typeof(md) = k_{this}\ q_{this},\ \overline{k_p}\ \overline{q_p} \rightarrow t_{ret} \qquad \Gamma = (this : k_{this}\ q_{this},\ \overline{p : k_p\ q_p},\ \overline{y : k_{local}\ q_{local}}) \qquad \Gamma \vdash s \qquad \Gamma(z) <: t_{ret} \qquad \text{Return type is polyimmutable => "this" is initialized polyimmutable} \vee \text{"this" is underinitialized } \_}{\vdash md \text{ is OK}} \text{(WF-METH)}$$

$$\frac{\vdash_C \bar{f} \text{ is OK} \qquad \vdash_C kd \text{ is OK} \qquad \vdash \overline{\overline{md}} \text{ is OK}}{\vdash C \text{ is OK}} \text{(WF-CLASS)}$$

Figure 4 Well-formdness typing