Nestmates Invocation Changes

Created by Karen Kinnear, last modified just a moment ago

Invokeinterface Selection Changes:

JVMS 6.5 Invokeinterface changes

Let *C* be the class of *objectref*. ~~The actual method to be invoked is selected by the following lookup procedure:~~ The selected method (5.4.5) for class *C* of an invocation of the referenced method is the actual method to be invoked.

1. ~~If *C* contains a declaration for an instance method with the same name and descriptor as the resolved method, then it is the method to be invoked.~~
2. ~~Otherwise, if *C* has a superclass, a search for a declaration of an instance method with the same name and descriptor as the resolved method is performed, starting with the direct superclass of *C* and continuing with the direct superclass of that class, and so forth, until a match is found or no further superclasses exist. If a match is found, then it is the method to be invoked.~~
3. ~~Otherwise, if there is exactly one maximally-specific method (5.4.3.3) in the superinterfaces of *C* that matches the resolved method's name and descriptor and is not `abstract`, then it is the method to be invoked~~...

Linking Exceptions

Otherwise, if the resolved method is `static` ~~or `private`~~, the `invokeinterface` instruction throws an `IncompatibleClassChangeError`.

5.4.5 Selection new section:

Where an `invokevirtual` or an `invokeinterface` invocation refers to a resolved method *mR*, the *selected method* of the invocation for an instance of a class or interface *C* is determined as follows:

If *mR* is marked `ACC_PRIVATE`, then it is the selected method.

Otherwise, the selected method is determined by the following lookup procedure:

1. If *C* contains a declaration for an instance method *m* that can override the resolved method, then *m* is the selected method.
2. Otherwise, if *C* has a superclass, a search for a declaration of an instance method that can override the resolved method is performed, starting with the direct superclass of *C* and continuing with the direct superclass of that class, and so forth, until a method is found or no further superclasses exist. If a method is found, it is the selected method.
3. Otherwise, if there is exactly one maximally-specific method (5.4.3.3) in the superinterfaces of *C* that matches the resolved method's name and descriptor and is not `abstract`, then it is the selected method.

> Note that any maximally-specific method selected in this step can override *mR*; there is no need to check this explicitly.

While *C* will typically be a class, it may be an interface when these rules are applied during preparation (5.4.2).

Beyond improving presentation, this change has the following effects:

- Addresses JDK-8098577: if an `invokevirtual` resolves to an interface method, *C* or one of its superclasses may override the interface method.
- Prevents `private` methods from being selected by `invokeinterface`, unless the `private` method is also the referenced method.

InvokeInterface Selection Table

Summary: Invokeinterface has been modified to support private members as well as to be more consistent with invoke virtual by defining selection based on overriding, which does not change any success cases, and does change some IAE cases to become success cases.

Resolved Method X Selected Method		local C.m public	local C.m Prot/PP	C.m private	SuperC.m public	SuperC.m Prot/PP	SuperC.m Object.m private	Object.m pubic	Object.m Prot/PP	SuperJ.m Pub/def
local I.m private	old:ICCE->I.m new:I.m	-	-	-	-	-	-	-	-	-
local I.m public	-	C.m	old:IAE new: C.m	old: IAE new: skip	SuperC.m	old: IAE new: SuperC.m	old: IAE new: skip	Object.m	old: IAE new:Object.m	SuperJ.m
Object public m	-	C.m	old: IAE new:	old: IAE new:	SuperC.m	old: IAE new:	old: IAE new: skip	Object.m	--	Object.m

			C.m	skip		SuperC.m				
SuperI: public default	-	C.m	old: IAE new: C.m	old: IAE new: skip	SuperC.m	old: IAE new: SuperC.m	old: IAE new: skip	Object.m	old: IAE new: Object.m	max-specific
SuperI: public (abstract)	-	C.m	old: IAE new: C.m	old: IAE new: skip	SuperC.m	old: IAE new: SuperC.m	old: IAE new: skip	Object.m	old: IAE new: Object.m	SuperJ.m

Not in this table:

1. If the selected method in local, SuperC, Object or SuperJ is Abstract -> AME unchanged.
2. If there is more than one maximally-specific method -> ICCE unchanged
3. If there are zero maximally-specific methods -> AME unchanged

Invokevirtual Selection Changes (TODO) (8098577)

6.5 Invokevirtual

Let *C* be the class of *objectref*. ~~The actual method to be invoked is selected by the following lookup procedure:~~ The selected method (5.4.5) for class *C* of an invocation of the referenced method is the actual method to be invoked.

1. ~~If *C* contains a declaration for an instance method *m* that overrides (5.4.5) the resolved method, then *m* is the method to be invoked.~~
2. ~~Otherwise, if *C* has a superclass, a search for a declaration of an instance method that overrides the resolved method is performed, starting with the direct superclass of *C* and continuing with the direct superclass of that class, and so forth, until an overriding method is found or no further superclasses exist. If an overriding method is found, it is the method to be invoked.~~
3. ~~Otherwise, if there is exactly one maximally-specific method (5.4.3.3) in the superinterfaces of *C* that matches the resolved method's name and descriptor and is not `abstract`, then it is the method to be invoked.~~

JVMS 5.4.5 Overriding new section on common selection rules

Addresses JDK-8098577: if an `invokevirtual` resolves to an interface method, *C* or one of its superclasses may override the interface method.

Invokevirtual behavior changes

- If a resolved method was private and in the same class as the caller, prior to this change, I do not see a clear selection definition since overriding does not apply to private methods. Implementation kept resolved == selected which is now clearly spelled out in the specification.
    - One test case to try: SuperC.m private, C extends SuperC. SuperC invoke virtual C.m
        - theory: earlier implementation: would allow invocation of SuperC.m via C.m. I believe the new specification would also
        - Question: is it the case that this is intentional for invoke virtual of nestmate private members? i.e. that the referenced method and the resolved method do not need to be the same?
- Addresses JDK-8098577: if an `invokevirtual` resolves to an interface method, *C* or one of its superclasses may override the interface method.
    - Question for Dan: I don't see how this is handled any differently.

Preparation Changes:

5.4.2 Preparation (changes)

During preparation of a class or interface *C*, the Java Virtual Machine also imposes loading constraints (5.3.4). Let *L1* be the defining loader of *C*. For each instance method *m* declared in *C* that ~~overrides~~ can override (5.4.5) ~~a~~ an instance method declared in a superclass or superinterface *<D, L2>*, the Java Virtual Machine imposes the following loading constraints: ...

Furthermore, ~~if *C* implements a~~ for each instance method *m* declared in a superinterface *<I, L3>* of *C*, ~~but~~ if *C* does not itself declare ~~the method~~ an instance method that can override *m*, then ~~let *<D, L2>* be the superclass of *C* that declares the implementation of method *m* inherited by *C*. The~~ if the selected method (5.4.5) for class or interface *C* of an invocation of *m* is declared in a class or interface *<D, L2>*, the Java Virtual Machine imposes the following constraints: ...

Changes in Preparation relative to Selection Behavior:

Prior to nestmates, the hotspot implementation treated the second paragraph as if it said "if C implements an ACC_PUBLIC method". We did not perform loader constraints for cases in which the selected method would throw an exception, since the intention of preparation is loader constraint checking when creating a selection cache.

With nestmates, and the modification to use the term override to match the common selection description, this would also allow ACC_PROTECTED and package private methods to be selected, and therefore have their loader constraints checked.

TODO: Sanity check - we never override a class method with an interface method according to the selection rules and in practice, so the first paragraph should describe preparation of a class C not of an interface C. Discuss with Dan S again, ask Dan H/Tobi/Graham - I do not think this is true even for java.lang.Object methods.