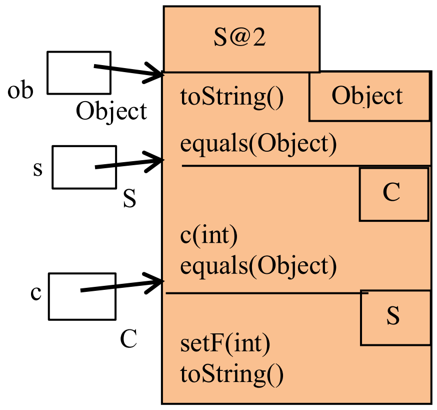
The overriding (or bottom-up) rule



Consider the object of class `S` on the right. Class `S` was declared as a subclass of class `C`. We show some (but obviously not all) of the methods in the three partitions.

Variables `ob`, `c`, and `s` were declared like this:

```
S s= new S(…);
C c= s;
Object ob= s;
```

Consider three possible calls on functions `toString` and on `equals`:

```
s.toString()        s.equals(some object)
c.toString()        c.equals (some object)
ob.tostring()       ob.equals (some object)
```

By the compile-time reference rule, all these calls are syntactically legal and will be compiled. We ask this question: At runtime, which method `toString` will be called, the one in partition `Object` or the one in partition `S`? The answer is given by this rule:

Overriding or bottom-up rule:

Let `p.m`(…) be a legal call on method `m`(…). To determine which method is called, start at the bottom of object `p` and search upward until the appropriate method `m` is found.

Applying this `rule`, *in all three cases*, method `toString` in partition `S` will be called. Similarly, in all three cases, function `equals` in partition `C` will be called.

This is an important point: at runtime, in determining which method is called when `ob.toString`() is called, *the type of variable* `ob` *does not matter*. What only matters is the object to which `ob` points.

Overriding or bottom-up rule for variables

The same rule applies for references to fields, like `s.f` (if there was a field `f`). But remember, we do *not* consider redeclaring fields. It can be done in Java, but we do not consider it and never do it. Thus, the object will have at most one field `f`.

Use of "super."

To the right is method `toString` in partition `S`. It returns the string "this is object S@2". The insertion of "**`super.`**" changes the bottom-up rule to start at the partition above partition `S`, so that method `toString` in partition `Object` is called. You know that in this case it returns "S@2".

toString in partition S

```
public String toString() {
  return "this is object " + super.toString();"
}
```

Thus, we have the "**super**." rule:

In any method `m` in a partition named `P`, the call **`super.`**`m`(…) calls the method `m` found by using the bottom-up rule starting at the partition above partition `P`.