

## Casting with interfaces

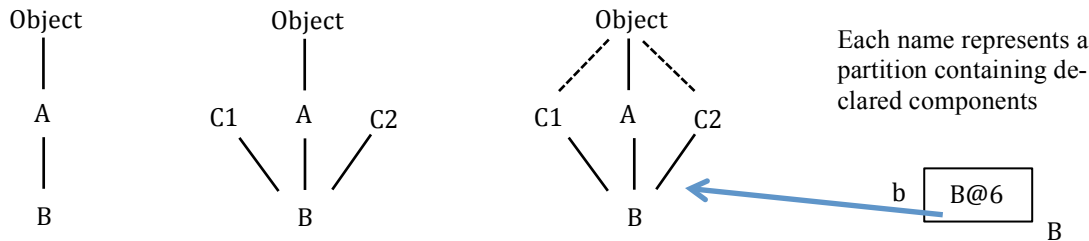
Here is a class A, two interfaces, and a class B that extends A and implements the two interfaces.

```
public class A { ... }
public interface C1 { ... }
public interface C2 { ... }
class B extends A implements C1, C2 { ... }
```

We show you what an object of class B looks like in three steps. First, we draw classes B, A, and Object, showing not the full partitions, as we have been doing, but only their names.

Second, since B implements C1, we draw a new line from B upward to C1, and the same for C2.

Third, we draw lines from C1 and C2 to Object, since, as we will see, the perspective of a C1 variable allows access to the methods in Object. Remember, class Object is the superest class of them all: any class or interface that does not explicitly extend something automatically extends Object. We use dashed lines, since they go from an interface upward to a class.



### Casting

We also show a variable `b` that contains a pointer to the object. This object can be cast to any of the classes and interfaces you see in the object, in any order, and to nothing else. For example, `b` can be cast to `A`, then to `C1`, and the result can be stored in variable `h`.

```
C1 h= (C1) (A) b;
```

Java will do upward casts automatically, as you know, but downward (or sideways casts) have to be done explicitly.

What method calls like `h.m(...)` are legal? Variable `h` has a `C1` perspective. The Java rule, as you know, is that `m` must be declared in `C1` or its superclass, `Object`. If `m` is not declared in `C1` or `Object`, the call is illegal and the program will not compile.

If `m(...)` is legal, which `m(...)` will be called at runtime? As always, the overriding one.

The call `h.equals(...)` is legal, since `equals` is declared in superest class `Object`. What one is called? Look first in `B`, then `A`, then `Object`.

### Interfaces may appear more than once in an object

You can skip this part and come back to it when you are more at ease with interfaces.

Let's add a new interface `C3` and a new class `D`. Note that interface `C3` extends interface `C1`. To the right, we draw an object of class `D` as well as a variable that points to it.

```
public interface C3 extends C1 { ... }
public class D extends B implements C3 { ... }
```

Interfaces `C1` appear twice in the object! If we do a cast

```
(C1) d
```

which `C1` is meant? Well, it doesn't matter. First, the same methods `m(...)` are available from both perspectives. Second, the overriding method `((C1) d).m(...)` is the same in both—the one in class `D`. In fact, it would be OK to have just one partition for `C1` in the hierarchy, with an upward line from each class or interface that extends it.

