

Objects and Equality

```
int a = 10;
int b = 10;

a == b; // true

Integer ia = new Integer(10);
Integer ib = new Integer(10);

ia == ib; // false, we need semantic equality here

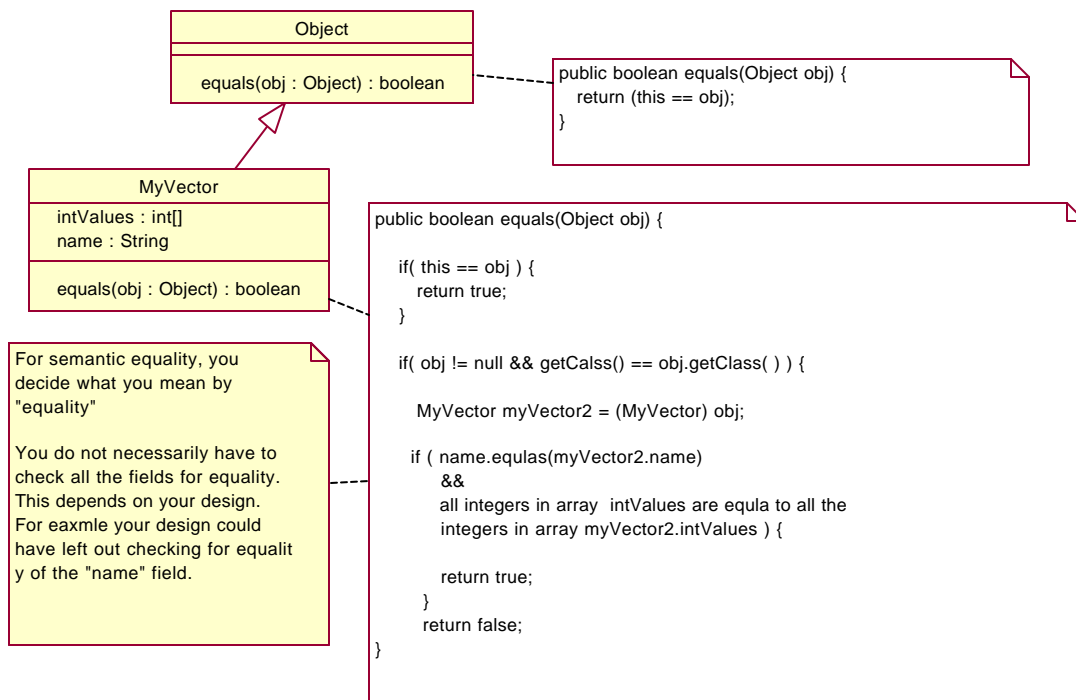
ia.equals(ib); // true, Class Integer contains an implementation of the equals method

// equals can only be used on references to objects, not on primitive types

int[] ints = new int[2];
ints[0] = 1;
ints[1] = 2;

MyArray arr1 = new MyArray(ints);
MyArray arr2 = new MyArray(ints);

arr1 == arr2; // false
arr1.equals(arr2); // false unless MyArray overrides the equals() method
```



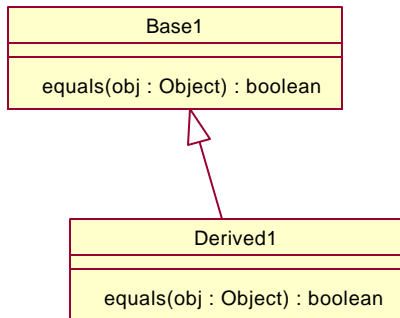
Be very careful when you implement equals method.

What would happen if you change your implementation of MyVector to store "name" as a StringBuffer, instead of a String?

Code above uses name.equals(myVector2.name), i.e. the overridden equals() method in class String. This gives a semantically correct result.

If name is a StringBuffer() then the default equals() in class Object is invoked, since StringBuffer() class does not override the equals() method. The default equals() is not implemented to check for semantic equality.

calling super.equals()

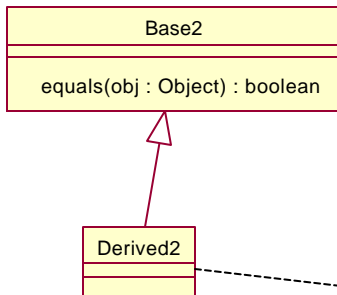


```
public boolean equals(Object obj) {
    if( this == obj ) {
        return true;
    }
    if( obj != null && getClass() == obj.getClass() && super.equals(obj) ) {
        super.
    }
    if ( name.equals(myVector2.name)
        &&
        all integers in array intValues are equal to all the
        integers in array myVector2.intValues ) {
        return true;
    }
    return false;
}
```

Do you want the possibility of derived class objects to equal base class objects?

This could make sense if the derived class has no additional attributes compared with the base class.

using instanceof operator



Base class implements equals() using the "instanceof" operator, not getClass() method.

```
public boolean equals(Object obj) {
    if( obj instanceof Base2 ) {
        ...
    }
}
```

Derived class does not implement equals()

Reflection in Java (Intro)

Reflection:

Every class and interface is represented by a **Class object**.
The Class class is the starting point of reflection.

Examples of getting an instance of Class:

```
Class classObject1 = object.getClass();
```

```
Class classObject2 = Double.class;  
Class classObject3 = MyArray.class;
```

```
String className = "..."; // fully qualified name  
Class classObject4 = Class.forName(className); // forName  
// is a static method
```