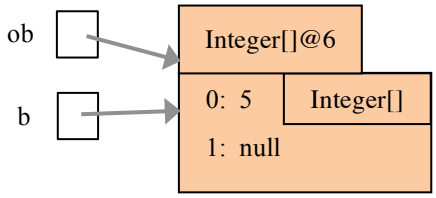
Why ArrayList<Integer> is not a subclass of ArrayList<Object>

Integer[] is a subclass of Object[]

Since the beginnings of Java, Integer[] has been a subclass of Object[]. Thus, the following code is syntactically correct and will compile.

```
Integer[] b= new Integer[2];
b[0]= 5;
Object[] ob= b;
ob[1]= "xyz";
```



How is this code executed?

1. A (pointer to a) new Integer array of length 2 is stored in b. Each array element contains null.
2. An Integer object containing 5 is stored in b[0].
3. The pointer in b is cast to Object[] and stored in ob. This is a widening cast, so it is done automatically.
4. The string "xyz" is cast to class Object and an attempt is made to store it in ob[1]. This would be a disaster if it were allowed, because the elements of array b have to be Integers. And indeed, execution causes an ArrayStoreException.

Thus, the Java designers allowed Integer[] to be a subclass of Object[] but put in special tests to ensure that only Integers would be stored in elements of an Integer array. We urge you to copy the above code into a Java procedure and see that it compiles but, at runtime, causes an ArrayStoreException.

ArrayList<Integer> is not a subclass of ArrayList<Object>

Now consider the equivalent code using an ArrayList:

```
ArrayList<Integer> c= new ArrayList<Integer>();
c.add(5);
ArrayList<Object> oc= c;      // Syntactically incorrect —see point 1 below.
c.add("xyz");                 // Syntactically incorrect —see point 2 below.
```

This code does not compile, for two reasons:

1. ArrayList<Integer> is *not* a subclass of ArrayList<Object>. The designers chose not to allow this.
2. Method c.add on the last line is not defined for a String argument.

If they *did* allow ArrayList<Integer> to be a subclass of ArrayList<Object>, then the call c.add("xyz"); would also be syntactically correct.

The designers of Java did not allow ArrayList<Integer> to be a subclass of ArrayList<Object> because they could not figure out a way to throw a “StoreException” whenever an attempt was made to store a value of the wrong class, as was done in the case of arrays. They could do it for ArrayLists, but not for all the ways in which a generic class could be written. There is no simple, adequate way to define this.