Function equals

Equals in class Object

As you know, function `equals`, defined in class object, tests whether two objects are indeed the same object. It can be written as follows:

```
/**  = "This object and object ob are the same object". *
public boolean equals(Object ob) {
    return this == ob;
}
```

For example, for two variables c1 and c2 that you see here, `c1.equals(c1)` is true, and `c1.equals(c2)` is false.

Each class C (say) can override function equals. Here is its general specification:

```
/** = “Object ob is of class C and equals this object */
public boolean equals(Object ob)
```

There are two important points to note here. First, in order to override function equals, the parameter must have type `Object`. Second, the specification says that `ob` is of class C, so this property must be checked in the method body. For example, suppose `c1` contains an object of class C. Then, the call

```
c1.equals(new Integer(5))
```

must yield **`false`**, because the class of the argument is not C.

Function equals should be an equality relation

Now, the writer of function equals gets to decide what “object `ob` equals this object” means. The convention is that this is a real equality —provided `c1`, `c2`, and `c3` are objects of the same class, three properties hold:

Reflexive: `c1.equals(c1)` is true.

Symmetric: `c1.equals(c2)` and `c2.equals(c1)` yield the same value.

Transitive: If `c1.equals(c2)` and `c2.equals(c3)` are true, then so is `c1.equals(c3)`

Now, if you haven’t been told that equality has these properties, don’t worry. You don’t have to check that these properties hold for your function `equals`. If you just define `equals` using common sense, then it should have these properties.

Function `equals` generally checks for equality of some or all fields of the two objects, but it should be specified in an abstract way, in terms of the meaning of the class and not the implementation with fields. For example, equality on `String` objects is defined by

The two strings (sequences of characters) contained in the objects are equal

We don’t know what fields are used in class `String`; we know only that, somehow, each instance contains a sequence of characters, and the equality of these sequences is being tested in function equality.

So, write your specification in terms of the meaning of the class and not in terms of its fields.

An example of writing a function equals

We write a function `equals` for class `Chapter`, where a chapter has a number, a title, and a previous chapter. Let’s write it step by step. First the specification. Now the header. And we stub in a return statement so that we can compile.

We write the function body. Look at the spec; the first thing it says is that ob must have class `Chapter`. So, let’s put in an if-statement to check it and return false if it is not a `Chapter`. Note that if `ob` is false, **`instanceof`** will evaluate to **`false`**, so that the function returns **false**.

Do you see how we refer to the method specification continually when writing the method body? If we hadn’t written the spec first, we would not have been able to do this so easily and, quite likely, we would have made a mistake. Write method specs before writing method bodies.

Function equals

Now, we know that c is a Chapter, so let's define a local variable, cast ob to Chapter, and store the result is c. This is so that the fields of instance Chapter, or c, can be referenced.

Now comes a single return statement that tests each of the required items. We begin with the chapter number —for emphasis here, we put in keyword **this**, although it is not necessary because field number is accessible anyway. Since field number has type **int**, we use equality ==. Now comes the chapter title. Since it is a String, we use function equals class String —using ==would be a mistake. Finally, we have the previous chapter. Note that we use equality ==, and not function equals! This is important. We want to test whether the two previous objects are the *same* object, not that they are equal in terms of having equal values in the objects.

And that's it! It's rather simple. All equals functions will have this form. First, make sure the parameter has the right class. Then, cast it to that class. Finally, make the equality test.

```
/** = "ob is of class Chapter and has the same
       chapter number, chapter title, and
       previous chapter as this object". */
public boolean equals(Object ob) {
    if (!(ob instanceof Chapter))
        return false;

    Chapter c= (Chapter) ob;

    return this.number == c.number &&
           this.equals(c.title) &&
           this.prev == c.prev;
}
```