

Re: Question on Effective Java Item 27

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2006-04/msg00199.html>

- *From:* Andrew McDonagh <news@xxxxxxxx>
 - *Date:* Thu, 20 Apr 2006 17:34:50 +0100
-

S Perryman wrote:

<hforco@xxxxxxxxxxxxxxxx> wrote in message
news:1145548622.568432.125550@xx

S Perryman wrote:

This is a common problem, specifically that one is attempting to use one value to convey multiple meanings. So either :

1. define multiple parameters, one to retrieve the outcome of the search, and the other for the value.

By 'multiple parameters' do you mean multiple methods – or are you referring to 'out' parameters?

The latter.

Java doesn't have out parameters.

I know.

But I wanted to explain both solutions as the problem is so common.

2. define a result type that holds both parameters

Re: Question on Effective Java Item 27

An example of 2 is :

```
class FDResult
{
    boolean found ;
    byte[] value ;
}
```

```
FDResult findData(Object key)
// post : RESULT.found XOR (RESULT.value == null)
```

I like this ... particularly as using the two-method approach (isExistingKey, followed by getValue) would mean that look-up of the key is done twice.

Correct.

Although I appreciate that this is more explicit than using a null return value, do you not agree that as long as it was documented that null return value meant the key was not found, that it would still work in the same way?

If it is documented so, then the intent is clear.

There is nothing fundamentally different between the following:

```
byte[] value = findData(key);
```

```
if (value == null) {
    // key not found ....
} else {
    // do something with value
}
```

Re: Question on Effective Java Item 27

and this

```
FDResult result = findData(key);
```

```
if (result.found) {  
    byte[] value = result.value;  
    // do something with value  
} else {  
    // key not found  
}
```

... other than the fact that the second example is much clearer.

But in general, there are cases where the value required is not an object reference, but a real value. For example :

```
int findData(Object key)
```

What are you going to use as the 'not found' value ??

Or will you rewrite the interface to be :

```
java.lang.Integer findData(Object key)
```

so that you can use the null value as 'not found' ??

Because I have programmed in different prog langs, I try to have a consistent approach for a problem regardless of the prog lang.

So here it is multiple out parameters, or a suitable result type.

But never null = not found.

null for not found is common in Java though

e.g.

[http://java.sun.com/j2se/1.4.2/docs/api/java/util/HashMap.html#get\(java.lang.Object\)](http://java.sun.com/j2se/1.4.2/docs/api/java/util/HashMap.html#get(java.lang.Object))

HashMap

```
public Object get(Object key)
```

Re: Question on Effective Java Item 27

Returns the value to which the specified key is mapped in this identity hash map, or null if the map contains no mapping for this key. A return value of null does not necessarily indicate that the map contains no mapping for the key; it is also possible that the map explicitly maps the key to null. The `containsKey` method may be used to distinguish these two cases.