Null Object design pattern troubles with behaviors

I want share some of my thoughts and ask some questions Null object design pattern supposed to implement class with neutral behavior. So, if i need implement interface:

```
public interface IProduct
{
    double Cost();
    string Name();
}
```

probably, i will use next implementation:

```
public class NullProduct : IProduct
{
    public double Cost()
    {
        return 0.0;
    }

    public string Name()
    {
        return String.Empty;
    }
}
```

Fine.

But, which strategy shall i use, when i need implement next interface:

```
public interface IProduct
{
    //other methods...
    bool IsTasty();
}
```

IsTasty - there no "neutral" behavior. How should i implement it in NullProduct class? Return true or false. No so clear.

Worse, if interface have some properties:

```
public interface IProduct
{
    //other methods...
    int Price{get;set;}
}
```

and implementation:

```
public class NullProduct : IProduct
{
    //other methods...
    int Price
    {
        get {return 0;}
        set {/*nothing*/}
    }
}
```

Why? Because of, if some user will get the object, like here:

```
    IProduct prod = GetSomeObject(); //GetSomeObject returns NullProduct
```

and user try do:

```
    prod.Price = 8;
    Console.WriteLine(prod.Price);
```

user will get strange result. User remembers, he stets 8 dollar price, but now price get 0. Inflation?

In UnitTest same problem.

Now, i do not want violate OCP by asking object if it null object.

How do you solve the problem?

design-patterns null-object-pattern

asked Dec 26 '12 at 6:47

zzfima

833 1 6 17

2 Answers

If you do not have a "neutral" return result, you should "invent" one, and use it in the null object:

```
enum Tasty {
    Yes, No, Unknown
}

public interface IProduct {
    Tasty IsTasty();
}
```

Writing to a null object is nearly inevitably a programming error, so the implementation of `Price` should go like this:

```
public class NullProduct : IProduct {
    //other methods...
    int Price {
        get {return 0;}
        set { throw new InvalidOperationException(); }
    }
}
```

answered Dec 26 '12 at 6:53
dasblinkenlight
588k 54 721 1143

Thanks! 1. IsTasty designed to be Boolean. For example it could be interface i can not change. 2. Does "throw new InvalidOperationException()" says, that i can use NullObject DP only for interfaces that have not "set" accessor? thanks – zzfima Dec 26 '12 at 9:02

1 @zzfima If you cannot change the interface to allow for a neutral behavior, such as "unknown" taste, then you cannot use the pattern, because by the very definition "a null object is an object with defined neutral behavior". If neither `Tasty=true` nor `Tasty=false` is neutral in the context of your object, then you cannot use the null object pattern. Another way to add "neutral" to `Taste` is to change the type to `bool?` , but it may be impossible because someone else controls the interface. – dasblinkenlight Dec 26 '12 at 15:09

@zzfima You can use null objects with mutable interfaces only with the understanding that such objects must not be edited (even though the interface allows it). A typical thing to do in such cases is to add a `bool IsReadOnly` property to the interface, and make the Null Object implementation always return `true` for `IsReadOnly` . – dasblinkenlight Dec 26 '12 at 15:11

It is possible to "fully componentize" the NullObject design pattern into **a generic reusable implementation** so you don't need to worry about handling all possible details in each case (as long as what you need to Nullify is an interface type). Note that it will also handle the case where your interface function returns another User Defined Type. In this case it will create another NullObject instance and return it, so you get deep NullObject instances consistently.

This is a generic reusable implementation NullObject.java and here you can see how it can be used TestNullObject.java

answered Dec 26 '12 at 17:04

SkyWalker
5,008 7 40 91