Null Object Design Pattern

A null object is also known as a Stub, an Active Nothing or an Active Null. It helps our program logic to get rid of null checks where ever possible. We all know, that we can not call methods on a null reference as it results in a NullReferenceException. The null object pattern provides a non-functional object in place of a null reference and therefore allows methods to be called on it.

Gaurav Gahlot Jul 29 2017

Introduction

In almost every method we write, irrespective of the programming language, we always have checks like the following one.

```
if (objectVariable == null)
    return;
// do some work with objectVariable
```

We end up adding so many null checks in our methods that it gets quite hard to figure out what we are supposed to do in our method. This makes our code look ugly and hard to read.

Scenario

from a MobileRepository using a method named GetMobileByName (*mobileNam* is mobile that we wish to operate on. A code implementing the above requirement would look something like the following.

```
01. class Program
02. {
03.     static void Main(string[] args)
04.     {
05.         var mobileRepository = new MobileRepository();
06.         IMobile mobile = mobileRepository.GetMobileByName("sony");
07.         mobile.TurnOn();
08.         mobile.TurnOff();
09.      }
10. }
```

Now, in this code, as we are not having a null check for the object returned by MobileRepository, we might get a NullreferenceException for some invalid input to the method GetMobileByName. In this case, we can solve this problem by

But is it really a good solution? What if the GetMobileByName is being used in several methods? Would it be feasible to add a null check in every method? Well, a better alternative is to implement Null Object pattern.

What is Null Object pattern?

A null object is also known as a Stub, an Active Nothing or an Active Null. It helps our program logic to get rid of null checks where ever possible. We all know, that we can not call methods on a null reference as it results in a NullReferenceException. The null object pattern provides a *non-functional* object in place of a null reference and therefore allows methods to be called on it.

So, for some invalid input to the method GetMobileByName, our MobileRepository would return an instantiated, yet null, IMobile object in place of a null reference. Unlike any other Mobile object, when we perform any operation on this object, nothing would happen.

Implementation

IMobile object in place of a null reference, for any invalid input. In order to achiev cr implementing from the IMobile interface. The object of this class would be our null object implementation. Please refer to the following code for the same.

```
01. public interface IMobile
02. {
03.     void TurnOn();
04.     void TurnOff();
05. }
06.
07. //mobile type implementing IMobile interface
08. public class SamsungGalaxy : IMobile
09. {
10.     public void TurnOff()
11.     {
12.         Console.WriteLine("\nSamsung Galaxy Turned OFF!");
13.     }
14.
15.     public void TurnOn()
16.     {
17.         Console.WriteLine("\nSamsung Galaxy Turned ON!");
18.     }
19. }
20.
21. //our null object class implementing IMobile interface as a singleton
22. public class NullMobile : IMobile
23. {
24.     private static NullMobile _instance;
25.     private NullMobile()
26.     { }
27.
28.     public static NullMobile Instance
29.     {
30.         get {
31.             if (_instance == null)
32.                 return new NullMobile();
33.             return _instance;
```

```
    //do nothing methods
    public void TurnOff()
    { }

    public void TurnOn()
    { }
}
```

Our repository can return different types of mobiles like SamsungGalaxy, AppleIPhone, and SonyXperia. All the types implement the IMobile interface. Our null object class, NullMobile is a classic Singleton and implements from IMobile interface. In the above code, we can see that, the NullMobile class like other types, implements the methods TurnOn and TurnOff. However, the method implementations do nothing.

Now, that we have our null object ready, it's time to use it at the right place. We do not want to have any null check for the type returned from the repository in the main logic. Therefore, our MobileRepository shall return a NullMobile instance in place of a

```
public class MobileRepository
{
    public IMobile GetMobileByName(string mobileName)
    {
        IMobile mobile = NullMobile.Instance;
        switch (mobileName)
        {
            case "sony":
                mobile = new SonyXperia();
                break;

            case "apple":
                mobile = new AppleIPhone();
                break;

            case "samsung":
                mobile = new SamsungGalaxy();
                break;
        }
```

We have first created a variable of type IMobile and initialized it with an instance of NullMobile. If the mobileName does not match a case in the switch statement, the NullMobile object is returned by the GetMobileByName method. In our main logic, when we try to call methods on this object we will not get any exception and hence the work flow will not break. However, the methods will do nothing.

Summary

- The null object pattern helps us to write a clean code avoiding null checks where ever possible.
- Using the null object pattern, the callers do not have to care whether they have a null object or a real object.
- It is not possible to implement null object pattern in every scenario. Sometimes, it is likely to return a null reference and perform some null checks.
- In above code sample, we are using an null object implementing an interface. However, we can have a quite similar approach with Abstract base class also.

Null pattern is helpful in situations where we want to return an object of the expected type, yet do nothing.

- It is important to note that unless developers are aware that the null object implementation exists, they may still do null checks.

I hope this helps you get a basic understanding of the null object pattern and how it helps you write a clean code. It's always great to have feedback from the readers. Your valuable feedback, questions, or comments about this article are always welcome.

Design Patterns NULL Object

Docker, Kubernetes and ASP.Net Core. He is an active contributor and has participated in many dev... Read more

https://www.quickdevnotes.com/ https://www.youtube.com/GauravGahlot

194 **387.4k** **2**

5 **3**

Type your comment here and press Enter Key (Minimum 10 characters)

Nice post, thanks. a minor fix: _instance is never assigned in NullMobile

Jawaharlal Rajan

1775 **2** **0**

Apr 06, 2019

0 0 Reply

Nice post! Thank you! But.. what if I want to have, for example, a method in the interface that returns the number of pixels in the display, and call it from an external method to calculate something? In this case I should return 0 (or -1, or whatever) as result and ...

cus

C# Corner
Guide To Machine Learning

1659 118 0

ASK A QUESTION CONTRIBUTE

Thanks Davide! I think that's one use case in which this pattern can be used. And returning (0,0) as the default does make sense in your example.

Gaurav Gahlot

194 9.4k 387.4k

Aug 01, 2017

0

TRENDING UP

01 Most Popular Front End JavaScript Framework In The World

02 Angular 7 Routing And Preserving Trailing Slash In URL

Top 10 JavaScript Frameworks In The World

04 .NET Core For .NET Developers

05 Microservices Using ASP.NET Core

06 Implement CRUD Operations With Sorting, Searching And Paging Using EF Core In ASP.NET Core

07 For Vs Foreach In C#

08 Building High Performance Back End (SQL Server)

09 All About C# Immutable Classes

10 How To Implement Authentication Using Identity Model In ASP.NET Core