Null object Design Pattern

The Null object pattern is a design pattern that simplifies the use of dependencies that can be undefined. This is achieved by using instances of a concrete class that implements a known interface, instead of null references.
We create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implemention of this class and will be used seemlessly where we need to check null value.

UML Diagram for Null object Design pattern

Design components

- **Client :** This class has a dependency that may or may not be required. Where no functionality is required in the dependency, it will execute the methods of a null object.
- **DependencyBase :** This abstract class is the base class for the various available dependencies that the Client may use. This is also the base class for the null object class. Where the base class provides no shared functionality, it may be replaced with an interface.
- **Dependency :** This class is a functional dependency that may be used by the Client.
- **NullObject :** This is the null object class that can be used as a dependency by the Client. It contains no functionality but implements all of the members defined by the DependencyBase abstract class.

Let's see an example of Null object design pattern.

```
// Java program to illustrate Null
// Object Design Pattern

abstract class Emp
{
    protected String name;
    public abstract boolean isNull();
    public abstract String getName();
}

class Coder extends Emp
{
    public Coder(String name)
    {
        this.name = name;
    }
    @Override
    public String getName()
    {
        return name;
    }
    @Override
    public boolean isNull()
    {
        return false;
    }
}

class NoClient extends Emp
{
    @Override
    public String getName()
    {
        return "Not Available";
    }
```

```
    @Override
    public boolean isNull()
    {
        return true;
    }
}

class EmpData
{

    public static final String[] names = {"Lokesh", "Kushagra", "Vikram"};
    public static Emp getClient(String name)
    {
        for (int i = 0; i < names.length; i++)
        {
            if (names[i].equalsIgnoreCase(name))
            {
                return new Coder(name);
            }
        }
        return new NoClient();
    }
}

public class Main
{
    public static void main(String[] args)
    {
        Emp emp1 = EmpData.getClient("Lokesh");
        Emp emp2 = EmpData.getClient("Kushagra");
        Emp emp3 = EmpData.getClient("Vikram");
        Emp emp4 = EmpData.getClient("Rishabh");


        System.out.println(emp1.getName());
        System.out.println(emp2.getName());
        System.out.println(emp3.getName());
        System.out.println(emp4.getName());
    }
}
```

Output:

```
Lokesh
Kushagra
Vikram
Not Available
```

Advantages :

- It defines class hierarchies consisting of real objects and null objects. Null objects can be used in place of real objects when the object is expected to do nothing. Whenever client code expects a real object, it can also take a null object.
- Also makes the client code simple. Clients can treat real collaborators and null collaborators uniformly. Clients normally don't know whether they're dealing with a real or a null collaborator. This simplifies client code, because it avoids having to write testing code which handles the null collaborator specially.

Disadvantages :

- Can be difficult to implement if various clients do not agree on how the null object should do nothing as when your AbstractObject interface is not well defined.
- Can necessitate creating a new NullObject class for every new AbstractObject class.

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Design Patterns | Set 1 (Introduction)

Design Patterns | Set 2 (Factory Method)

Command Pattern

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern

Article Tags : Design Pattern

Be the First to upvote.

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org