Null Object Design Pattern

In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do nothing relationship. Such Null object can also be used to provide default behaviour in case data is not available.

In Null Object pattern, 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.

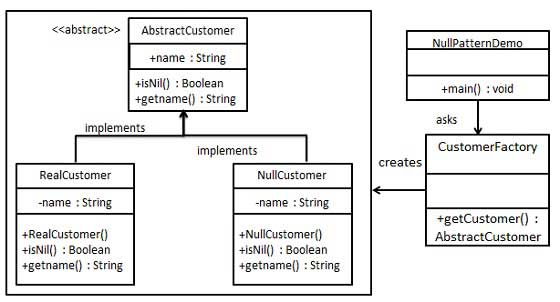
**Implementation**

We are going to create a *AbstractCustomer* abstract class defining opearations. Here the name of the customer and concrete classes extending the *AbstractCustomer* class. A factory class *CustomerFactory* is created to return either *RealCustomer* or *NullCustomer* objects based on the name of customer passed to it.

*NullPatternMain*, our demo class, will use *CustomerFactory* to demonstrate the use of Null Object pattern.

**The participants classes in this pattern are:**

* **AbstractClass** - defines abstract primitive operations that concrete implementations have to define.
* **RealClass** - a real implementation of the AbstractClass performing some real actions.
* **NullClass** - a implementation which do nothing of the abstract class, in order to provide a non-null object to the client.
* **MainClass** - the mainClass gets an implementation of the abstract class and uses it. It doesn't really care if the implementation is a null object or an real object since both of them are used in the same way.



**Example: Log System**

Let's say we need a logging framework in order to support the logging of an application. The framework must fulfill the following requirements:

* The destination of the output messages should be selected from a configuration file and it can be one of the following options: Log File, Standard Console or Log Disabled.
* Must be open for extension; new logging mechanism can be added without touching the existing code.

**Specific problems and implementation**

1. **Null Object and Factory**

The Null Object design pattern is more likely to be used in conjunction with the Factory pattern. The reason for this is obvious: A Concrete Classes need to be instantiated and then to be served to the client. The client uses the concrete class. The concrete class can be a Real Object or a Null Object.

1. **Null Object and Template Method**

The Template method design pattern need to define an abstract class that define the template and each concrete class implements the steps for the template. If there are cases when sometimes template is called and sometimes not then, in order to avoid the checking a Null Object can be use to implement a Concrete Template that does nothing.

1. **Removing old functionality**

The Null Object can be used to remove old functionality by replacing it with null objects. The big advantage is that the existing code doesn't need to be touched.

**Conclusion**

The Null Object Pattern is used to avoid special if blocks for do nothing code, by putting the “do nothing” code in the Null Object which becomes responsible for doing nothing. The client is not aware anymore if the real object or the null object is called so the 'if' section is removed from client implementation.