DESIGN PATTERNS ASSIGNMENT

NULL OBJECT

The Goal of a Null Object design pattern is to encapsulate the absence of an object by providing a substitutable alternative that offers suitable default do nothing behaviour. In short, a design where "nothing will come of nothing"

We use Null object where:

1. an object requires a collaborator. The Null Object pattern does not introduce this collaboration--it makes use of a collaboration that already exists
2. some collaborator instances should do nothing
3. you want to abstract the handling of null away from the client

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.

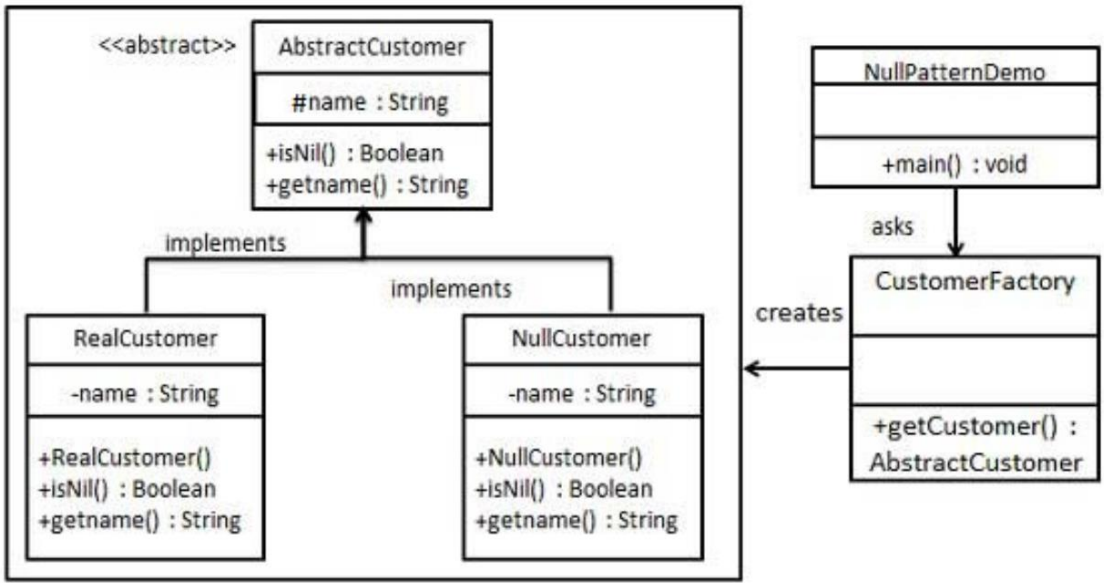
In Null Object pattern, we create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implementation of this class and will be used seamlessly where we need to check null value.

Structure:

- Client -
  - requires a collaborator.
- AbstractObject -
  - declares the interface for Client's collaborator
  - implements default behaviour for the interface common to all classes, as appropriate
- RealObject -
  - defines a concrete subclass of AbstractObject whose instances provide useful behaviour that Client expects
- NullObject -
  - provides an interface identical to AbstractObject's so that a null object can be substituted for a real object
  - implements its interface to do nothing. What exactly it means to do nothing depends on what sort of behaviour Client is expecting
  - when there is more than one way to do nothing, more than one NullObject class may be required

Implementation:

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. NullPatternDemo, the demo class, will use CustomerFactory to demonstrate the use of Null Object pattern.



Step 1

Create an abstract class.

AbstractCustomer.java

```
public abstract class AbstractCustomer {
   protected String name;
   public abstract boolean isNil();
   public abstract String getName();
}
```

Step 2

Create concrete classes extending the above class.

RealCustomer.java

```
public class RealCustomer extends AbstractCustomer {

   public RealCustomer(String name) {
      this.name = name;
   }

   @Override
   public String getName() {
      return name;
   }

   @Override
   public boolean isNil() {
      return false;
   }
}
```

NullCustomer.java

```
public class NullCustomer extends AbstractCustomer {

  @Override
  public String getName() {
    return "Not Available in Customer Database";
  }

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

Step 3

Create *CustomerFactory* Class.

CustomerFactory.java

```
public class CustomerFactory {

  public static final String[] names = {"Rob", "Joe", "Julie"};

  public static AbstractCustomer getCustomer(String name){

    for (int i = 0; i < names.length; i++) {
      if (names[i].equalsIgnoreCase(name)){
        return new RealCustomer(name);
      }
    }
    return new NullCustomer();
  }
}
```

Step 4

Use the *CustomerFactory* to get either *RealCustomer* or *NullCustomer* objects based on the name of customer passed to it.

NullPatternDemo.java

```
public class NullPatternDemo {
   public static void main(String[] args) {

      AbstractCustomer customer1 = CustomerFactory.getCustomer("Rob");
      AbstractCustomer customer2 = CustomerFactory.getCustomer("Bob");
      AbstractCustomer customer3 = CustomerFactory.getCustomer("Julie");
      AbstractCustomer customer4 = CustomerFactory.getCustomer("Laura");

      System.out.println("Customers");
      System.out.println(customer1.getName());
      System.out.println(customer2.getName());
      System.out.println(customer3.getName());
      System.out.println(customer4.getName());
   }
}
```

Step 5

Verify the output.

```
Customers
Rob
Not Available in Customer Database
Julie
Not Available in Customer Database
```