Design Patterns - Null Object 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.

在Null Object①设计模式中，一个Null Object替换对一个值为null的object的检查，而不是通过if语句来判断该值是不是为null。Null Object反映了一种do-nothing②的关系。这样的Null Object也可以用于提供默认行为，当数据不可用时。

在 Null Object 模式中，我们创建一个 抽象类声明了各种各样需要之执行的操作，创建具体的类继承这个抽象类，并且创建一个Null Object类来提供do-nothing的实现，同时也能在我们需要的时候帮我们检测null值。

译注：

1. Null Object 直译的话就是空对象，感觉不够优雅，而且不太符合它的实际意义，所以暂时使用英文单词来代替。这里的Null Object不是指对象的值为null（Object object = null）而是指，该对象表达的是Null的含义。我们人为的为null赋予具体的含义，让null代表一种特殊的数据状态。来使程序语义更加的明确，避免空值和null等数据值使得程序语义混乱，以及null值导致程序的崩溃等等。这种为null赋予具体含义的思想 在Google的java类库guava中有很好的实现。
2. 在本文中do-nothing 指，若对象的值为null时，将不做任何动作。

Implementation

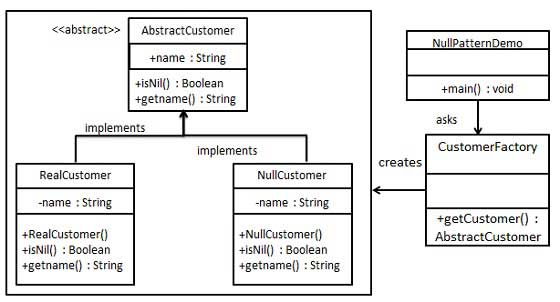
实现

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.

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

我们将创建一个抽象类AbstractCustomer，该类声明了两个方法以及记录客户端名字的属性值。创建具体的类集成该抽象类。工厂类CustomerFactory通过 客户端传递给它的名字来创建并返回RealCustomer或者NullCustomer的对象。

NullPatternDemo，我们的demo类，将使用CustomerFactory来展示如何使用NullObjectPattern。



Step 1

第一步

Create an abstract class.

创建抽象类

*AbstractCustomer.java*

public abstract class AbstractCustomer {

protected String name;

public abstract boolean isNil();

public abstract String getName();

}

Step 2

第二步

Create concrete classes extending the above class.

创建具体的类继承抽象类 AbstractCustomer.java

*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.

创建CustomerFactory类

*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

第四步

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

使用CustomerFactory类通过customer传递的name值来获取RealCustomer或者NullCustomer类的对象.

*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

第五步

Verify the output.

校验输出

Customers

Rob

Not Available in Customer Database

Julie

Not Available in Customer Database