



Design Patterns Tutorial

- ▣ [Design Patterns - Home](#)
- ▣ [Design Patterns - Overview](#)
- ▣ [Design Patterns - Factory Pattern](#)
- ▣ [Abstract Factory Pattern](#)
- ▣ [Design Patterns - Singleton Pattern](#)
- ▣ [Design Patterns - Builder Pattern](#)
- ▣ [Design Patterns - Prototype Pattern](#)
- ▣ [Design Patterns - Adapter Pattern](#)
- ▣ [Design Patterns - Bridge Pattern](#)
- ▣ [Design Patterns - Filter Pattern](#)
- ▣ [Design Patterns - Composite Pattern](#)
- ▣ [Design Patterns - Decorator Pattern](#)
- ▣ [Design Patterns - Facade Pattern](#)
- ▣ [Design Patterns - Flyweight Pattern](#)
- ▣ [Design Patterns - Proxy Pattern](#)
- ▣ [Chain of Responsibility Pattern](#)
- ▣ [Design Patterns - Command Pattern](#)
- ▣ [Design Patterns - Interpreter Pattern](#)

- ▣ Design Patterns - Iterator Pattern
- ▣ Design Patterns - Mediator Pattern
- ▣ Design Patterns - Memento Pattern
- ▣ Design Patterns - Observer Pattern
- ▣ Design Patterns - State Pattern
- ▣ Design Patterns - Null Object Pattern
- ▣ Design Patterns - Strategy Pattern
- ▣ Design Patterns - Template Pattern
- ▣ Design Patterns - Visitor Pattern
- ▣ Design Patterns - MVC Pattern
- ▣ Business Delegate Pattern
- ▣ Composite Entity Pattern
- ▣ Data Access Object Pattern
- ▣ Front Controller Pattern
- ▣ Intercepting Filter Pattern
- ▣ Service Locator Pattern
- ▣ Transfer Object Pattern

Design Patterns Resources

- ▣ Design Patterns - Questions/Answers
- ▣ Design Patterns - Quick Guide
- ▣ Design Patterns - Useful Resources
- ▣ Design Patterns - Discussion

Design Patterns - Null Object Pattern

Advertisements



Never Lose Your Keys Again

Only \$25



⬅ Previous Page

Next Page ➡

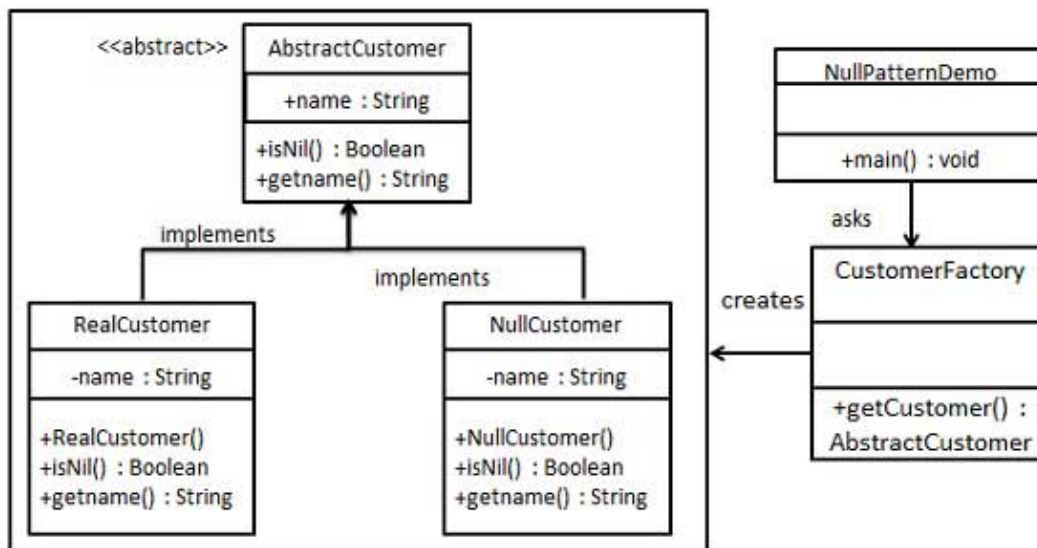
In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do nothing relationship. Such Null object can also be used to provide default behaviour in case data is not available.

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.

Implementation

We are going to create a *AbstractCustomer* abstract class defining operations. 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.



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
```

[⬅ Previous Page](#)

[Next Page ➡](#)

Advertisements





[Write for us](#) | [FAQ's](#) | [Helping](#) | [Contact](#)

© Copyright 2015. All Rights Reserved.

Enter email for newsletter

go