PATRÓN FACHADA

ING. SISTEMAS DE INFORMACION Y GESTION

"ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS"

INTEGRANTES:

- FLORES PUMA, LUIS
- ZAVALA DIAZ, ANIBAL
- JORGE DE LA CRUZ, JOSELYN

2017

¿QUÉ SON LOS PATRONES DE DISEÑO?

Cuando se desarrolla una aplicación software es frecuente encontrarse en la situación de tener que volver a resolver problemas similares a otros que ya hemos solucionado anteriormente, y debemos volver a hacerlo partiendo de cero una y otra y otra vez (incluso dentro del mismo proyecto).

Debido a ello y basándose en la programación orientada a objetos surgieron los patrones de diseño, donde cada uno de ellos define la solución para resolver un determinado problema, facilitando además la reutilización del código fuente.

Dependiendo de su finalidad pueden ser:

- **Patrones de creación**: utilizados para crear y configurar de clases y objetos.
- **Patrones estructurales**: su objetivo es desacoplar las interfaces e implementar clases y objetos. Crean grupos de objetos.
- **Patrones de comportamiento**: se centran en la interacción entre asociaciones de clases y objetos definiendo cómo se comunican entre sí.

	CREACIÓN	ESTRUCTURALES	COMPORTAMIENTO
CLASES	Factory Method		Interpreter Template Method
OBJETOS	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Al utilizarlos hemos de tener bien claro qué patrón utilizar en cada caso: si necesitamos realizarle más que unos cambios mínimos puede ser señal de que no sea el más adecuado en dichas circunstancias.

Otro concepto del que puedes oír hablar es el de **anti-patrones**, que hace referencia a los errores que comúnmente suelen ocurrir al intentar solucionar problemas conocidos.

No es necesario memorizar cómo funcionan todos los patrones, pero sí es importante saber de su existencia para así recurrir a ellos en caso necesario.

¿Qué es una fachada o facade en inglés?

Es un patrón de diseño que nos permite simplificar el interface de comunicación entre dos objetos A y B de tal forma que para el objeto A sea más sencillo interactuar con el objeto B. Supongamos que tenemos las siguientes clases:

Vamos a ver el código de Impresora (Java):

```
?
1       packagecom.genbetadev;
2
3       publicclassImpresora {
4
5
6           privateString tipoDocumento;
7           privateString hoja;
8           privatebooleancolor;
9           privateString texto;
10
11
12          publicStringgetTipoDocumento() {
13              returntipoDocumento;
14          }
15          publicvoidsetTipoDocumento(StringtipoDocumento) {
16              this.tipoDocumento = tipoDocumento;
17          }
18          publicString getHoja() {
19              returnhoja;
20          }
21          publicvoidsetHoja(String hoja) {
22              this.hoja = hoja;
23          }
24          publicbooleanisColor() {
25              returncolor;
26          }
27          publicvoidsetColor(booleancolor) {
28              this.color = color;
29          }
30          publicString getTexto() {
31              returntexto;
32          }
33          publicvoidsetTexto(String texto) {
34              this.texto = texto;
```

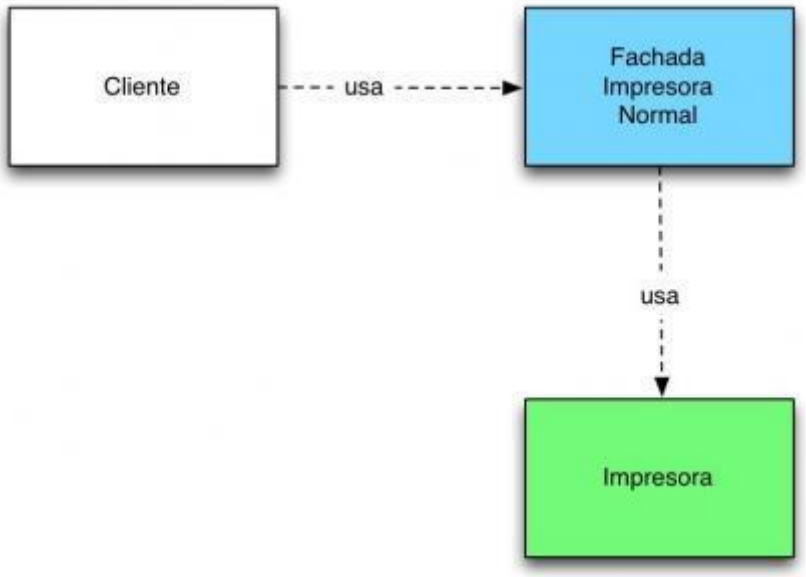
```
35          }
36
37          publicvoidimprimirDocumento() {
38
39              System.out.println("imprimiendo:"+ texto+ ",color:"
40              + color+",tipo:"+tipoDocumento+",hoja :"+hoja);
41          }
42      }
```

Se trata de una clase sencilla que imprime documentos en uno u otro formato. El código de la clase cliente nos ayudará a entender mejor su funcionamiento.

```
?
        packagecom.genbetadev;
1
2       publicclassPrincipalCliente {
3
4           publicstaticvoidmain(String[] args) {
5
6
7               Impresora i = newImpresora();
8               i.setHoja("a4");
9               i.setColor(true);
10              i.setTipoDocumento("pdf");
11              i.setTexto("texto 1");
12              i.imprimirDocumento();
13
14              Impresora i2 = newImpresora();
15              i2.setHoja("a4");
16              i2.setColor(true);
17              i2.setTipoDocumento("pdf");
18              i2.setTexto("texto 2");
19              i2.imprimirDocumento();
20
21              Impresora i3 = newImpresora();
22              i3.setHoja("a3");
23              i3.setColor(false);
24              i3.setTipoDocumento("excel");
25              i3.setTexto("texto 3");
26              i3.imprimirDocumento();
27          }
28
29      }
30
```

Como podemos ver la clase cliente se encarga de invocar a la impresora, y configurarla para después imprimir varios documentos .**Ahora bien prácticamente todos los documentos que escribimos tienen la misma estructura (formato A4, Color , PDF).** Estamos continuamente repitiendo código. Vamos a construir una nueva

clase **FachadaImpresoraNormal** que simplifique la impresión de documentos que sean los más habituales.



?

```
1  packagecom.genbetadev;
2
3  publicclassFachadaImpresoraNormal {
4
5      Impresora impresora;
6
7
8      publicFachadaImpresoraNormal(String texto) {
9          super();
10         impresora= newImpresora();
11         impresora.setColor(true);
12         impresora.setHoja("A4");
13         impresora.setTipoDocumento("PDF");
14         impresora.setTexto(texto);
15     }
16
17
18     publicvoidimprimir() {
19
20         impresora.imprimirDocumento();
21     }
22
23 }
```

De esta forma el cliente quedará mucho más sencillo :

```
?
1       packagecom.genbetadev;
2
3       publicclassPrincipalCliente2 {
4
5           publicstaticvoidmain(String[] args) {
6
7
8               FachadaImpresoraNormal fachada1= newFachadaImpresoraNormal("texto1");
9               fachada1.imprimir();
10
11              FachadaImpresoraNormal fachada2= newFachadaImpresoraNormal("texto2");
12              fachada2.imprimir();
13
14
15              Impresora i3 = newImpresora();
16              i3.setHoja("a4");
17              i3.setColor(true);
18              i3.setTipoDocumento("excel");
19              i3.setTexto("texto 3");
20              i3.imprimirDocumento();
21          }
22
23      }
```

Ventajas e inconvenientes

La principal ventaja del patrón fachada consiste en que para modificar las clases de los subsistemas, sólo hay que realizar cambios en la interfaz/fachada, y los clientes pueden permanecer ajenos a ello. Además, y como se mencionó anteriormente, los clientes no necesitan conocer las clases que hay tras dicha interfaz.

Como inconveniente, si se considera el caso de que varios clientes necesiten acceder a subconjuntos diferentes de la funcionalidad que provee el sistema, podrían acabar usando sólo una pequeña parte de la fachada, por lo que sería conveniente utilizar varias fachadas más específicas en lugar de una única global.

Usos conocidos (Problemas/Soluciones)

Problema: Un cliente necesita acceder a parte de la funcionalidad de un sistema más complejo.

- Definir una interfaz que permita acceder solamente a esa funcionalidad.

Problema: Existen grupos de tareas muy frecuentes para las que se puede crear código más sencillo y legible.

- Definir funcionalidad que agrupe estas tareas en funciones o métodos sencillos y claros.

Problema: Una biblioteca es difícilmente legible.

- Crear un intermediario más legible.

Problema: Dependencia entre el código del cliente y la parte interna de una biblioteca.

- Crear un intermediario y realizar llamadas a la biblioteca sólo o, sobre todo, a través de él.

Problema: Necesidad de acceder a un conjunto de APIs que pueden además tener un diseño no muy bueno.

- Crear una API intermedia, bien diseñada, que permita acceder a la funcionalidad de las demás.

Problema: Muchas clases cliente quieren usar varias clases servidoras, y deben saber cuál es exactamente la que le proporciona cada servicio. El sistema se volvería muy complejo, porque habría que relacionar todas las clases cliente con todas y cada una de las clases servidoras.

- Crear una o varias clases Facade, que implementen todos los servicios, de modo que o todos los clientes utilicen esa única clase, o que cada grupo de clientes use la fachada que mejor se ajuste a sus necesidades.

Segundo ejemplo:

Ejemplo demostrando el Patrón Fachada – Simplificar proceso de Crédito Financiero.

Tiene un paquete llamado Modulo

Clases:

- class Banco
- class Cliente
- class Credito
- class Façade
- class FacadeHipoteca
- class Prestamo

```
public class Banco {
    public boolean tieneFondos(Cliente c){
        System.out.println("El cliente tiene Fondos ");
        return true;
    }
}
```

```
public class Cliente {
    private String nombre;
    public Cliente(String nombre){
        this.nombre = nombre;
    }
    public String getNombre(){
      return nombre;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
```

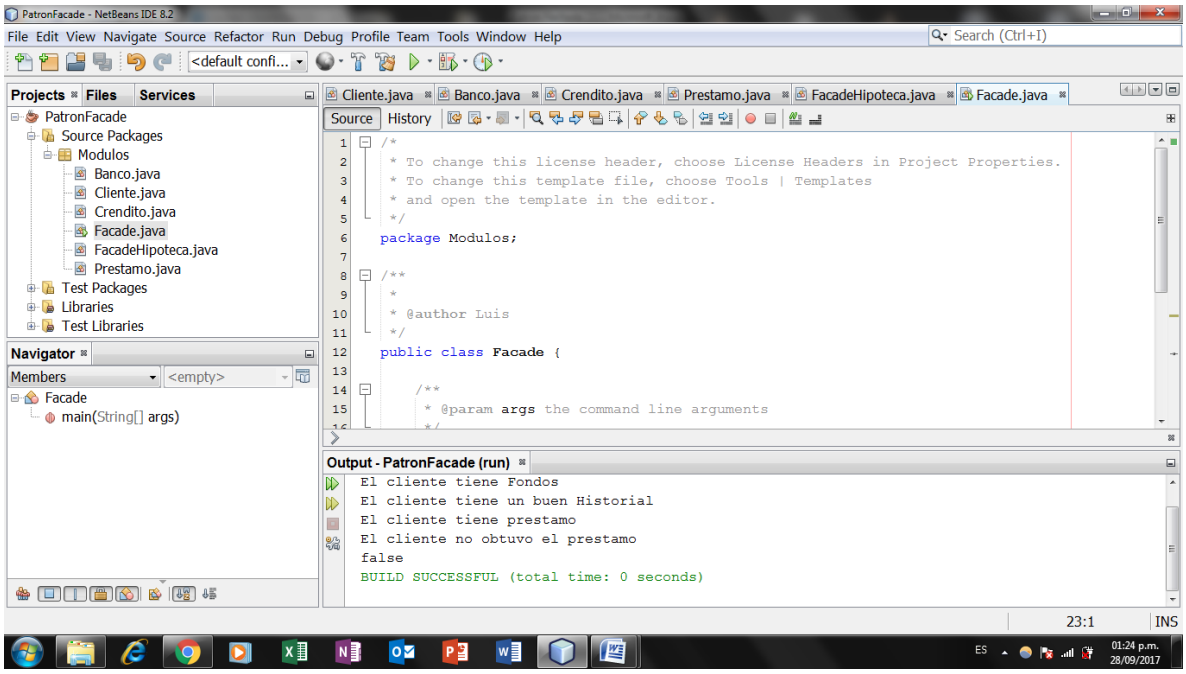
```
public class Credito {
    public boolean historialCorrecto(Cliente c){
        System.out.println("El cliente tiene un buen Historial");
        return true;
    }

}
```

```
public class Facade {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println(new FacadeHipoteca().esFiable(new Cliente("Laura")));
```

```
    }

}
-------------------------------------------------------------
public class FacadeHipoteca {
    public boolean esFiable(Cliente c){
        if((new Banco().tieneFondos(c) && (new Crendito().historialCorrecto(c)) && (new
Prestamo().tienePrestamo(c)))){
            System.out.println("El cliente obtuvo el prestamo");
            return true;
        }
        else {
            System.out.println("El cliente no obtuvo el préstamo");
            return false;
        }
    }
}
-------------------------------------------------------------
public class Prestamo {
    public boolean tienePrestamo(Cliente c){
        System.out.println("El cliente tiene préstamo ");
        return false;
    }
}
```

Resultado:

```
run:
El cliente tiene Fondos
El cliente tiene un buen Historial
El cliente tiene préstamo
El cliente no obtuvo el préstamo
false
BUILD SUCCESSFUL (total time: 0 seconds)
```


PatronFacade - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects Files Services
PatronFacade
Source Packages
Modulos
Banco.java
Cliente.java
Crendito.java
Facade.java
FacadeHipoteca.java
Prestamo.java
Test Packages
Libraries
Test Libraries
Navigator
Members
<empty>
Facade
main(String[] args)
Cliente.java Banco.java Crendito.java Prestamo.java FacadeHipoteca.java Facade.java
Source History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package Modulos;
7
8 /**
9 *
10 * @author Luis
11 */
12 public class Facade {
13
14 /**
15 * @param args the command line arguments
Output - PatronFacade (run)
El cliente tiene Fondos
El cliente tiene un buen Historial
El cliente tiene prestamo
El cliente no obtuvo el prestamo
false
BUILD SUCCESSFUL (total time: 0 seconds)
23:1
INS
ES
01:24 p.m.
28/09/2017