Fachada

Nombre:
Fachada

Clasificación del patrón:
Estructural

Intención:
Provee una interfaz unificada para un conjunto de interfaces en un subsistema. Fachada define un alto nivel de interfaces que hace el subsistema fácil de usar.

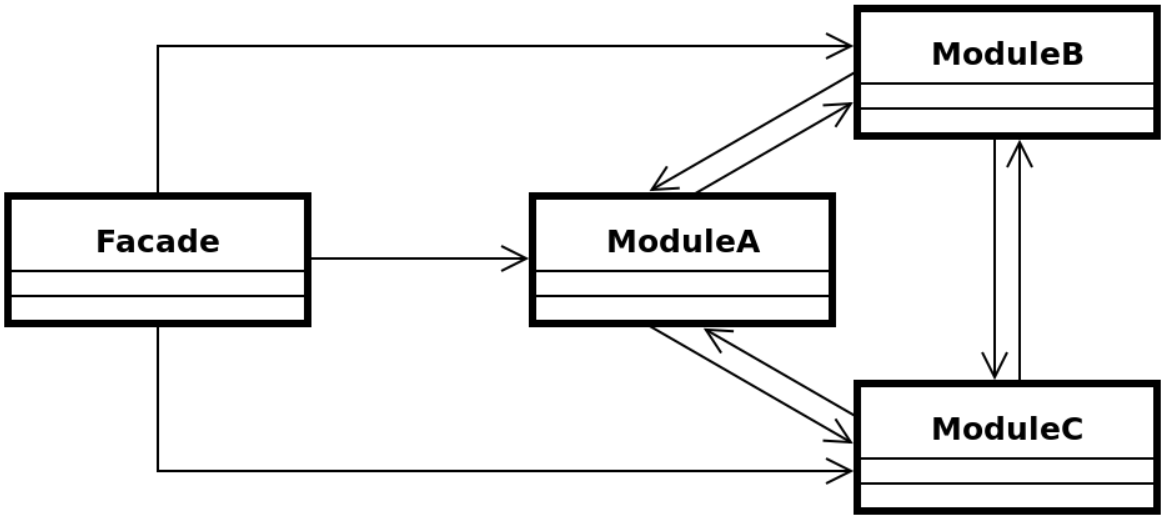
Otros nombres:
Facade

Motivación:
Uno de los objetivos comúnes en cuanto al desarrollo de software consiste en reducir y minimizar las comunicaciones entre subsistemas (acoplamiento), una forma de conseguirlo es precisamente usando Fachada para suministrar una interfaz simple ampliando la facilidad del sistema.

Aplicabilidad:
Se debe usar cuando:

- Se quiere avituallar una interfaz simple a un sistema complejo.
- Se busca hacer el sistema más fácil para nuevos usuarios, pero viejos usuarios quieran mantener su aplicación, se da un poderoso acceso al subsistema.
- Se piensa abastecer de nuevas y poderosas interfaces a diferentes niveles de usuario.
- Se reducen en gran medida las dependencias entre clientes y las clases que usan.

Estructura:



Participantes:

- Fachada: Delegan la petición del cliente a sus respectivos objetos en los subsistemas.
- Clase de subsistemas: Implementan la funcionalidad del sistema.

Colaboraciones:

- Los Clientes se comunican con el subsistema por medio de la Fachada que envía las peticiones a los objetos encargados.
- Los clientes que usan la Fachada no necesitan tener acceso directo a cada objeto con el que interactúan.

Ventajas:

- Aisla clases de las clases más complejas. Lo cual reduce el número de objetos con los que el cliente interactúa y también reduce la dependencia en la compilación de sistemas grandes.
- Existe un débil acoplamiento entre las clases del cliente y las clases del subsistema, porque las clases del subsistema pueden cambiar sin afectar las clases de los clientes.
- La Fachada no impide que los clientes usen las clases del subsistema.
- Para funciones de bajo nivel y estructuras de datos, se proporciona una interfaz orientada a objetos reduciendo los errores de programación.

Desventajas:

- El uso excesivo del patrón nos guía a un sistema poco funcional, con muchas capas y errores del funcionamiento.

Implementación:

- Se puede hacer una reducción extra del acoplamiento haciendo a la Fachada una clase abstracta que permita escoger entre diferentes implementaciones del subsistema.
- En Java se facilita la definición de clases privadas a un subsistema.

Código de ejemplo:

- Retirada de efectivo de un cajero automático.

```
1  public class Autentificacion{
2          /* ... */
3          public boolean leerTarjeta(){}
4          public String introducirClave(){}
5          public boolean comprobarClave(String clave){}
6          public Cuenta obtenerCuenta(){}
7          public void alFallar(){}
8  }
9
10 public class Cajero{
11         /* ... */
12         public int introducirCantidad(){}
13         public boolean tieneSaldo(int cantidad){}
14         public int expedirDinero{}
15         public String imprimirTicket(){}
16 }
17
18 public class Cuenta{
19         /* ... */
20         public double comprobarSaldoDisponible(){}
21         public boolean bloquearCuenta(){}
22         public boolean desbloquearCuenta{}
23         public void retirarSaldo(int cantidad){}
24         public boolean actualizarCuenta(){}
25         public void alFallar(){}
26 }
```

- Fachada:

```
public class FachadaCajero{
	private Autentificacion autentificacion = new Autentificacion();
	private Cajero cajero = new Cajero();
	private Cuenta cuenta = null;

	public void introducirCredenciales(){
		boolean tarjeta_correcta = autentificacion.leerTarjeta();
		if(tarjeta_correcta){
			String clave = autentificacion.introducirClave();
			boolean clave_correcta = autentificacion.comprobarClave(clave);
			if(clave_correcta){
				cuenta = autentificacion.obtenerCuenta();
				return;
			}
		}
		autentificacion.alFallar();
	}

	public void sacarDinero(){
		if(cuenta != null){
			int cantidad = cajero.introducirCantidad();
			int tiene_dinero = cajero.tieneSaldo(cantidad);
			if(tiene_dinero){
				boolean hay_saldo_suficiente = ((int)cuenta.comprobarSaldoDisponible()) >= cantidad;
				if(hay_saldo_suficiente){
					cuenta.bloquearCuenta();
					cuenta.retirarSaldo(cantidad);
					cuenta.actualizarCuenta();
					cuenta.desbloquearCuenta();

					cajero.expedirDinero();
					cajero.imprimirTicket();
				}
				else{
					cuenta.alFallar();
				}
			}
		}
```

- Clase principal:

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

	FachadaCajero cajero_automatico = new FachadaCajero();



	cajero_automatico.introducirCredenciales();

	cajero_automatico.sacarDinero();

}
```

Usos conocidos:

- Varias librerías de Software, como Font y Graphics en el caso de Java.

Patrones relacionados:

- Singletón

Bibliografía:

No específico. (No específico). GoF Design Patterns (Versión 2.1.0) [Aplicación móvil]. Descargado de: https://drive.google.com/file/d/0BywiVyFlIabXcVhGZlJBcnhWTkU/view.

Patrones de Diseño (XI): Patrones Estructurales - Facade [Página web]. (s.f.). Ubicación https://programacion.net/articulo/patrones_de_diseno_xi_patrones_estructurales_facade_1014.

Junta de Andalucía. (s.f). Fachada. Marco de Desarrollo de la Junta de Andalucía. http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/189.