Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México

Departamento de Computación
Diseño y Arquitectura de Software
Profesora Marlene O. Sánchez Escobar

“Selección de patrones a implementar”

Ángel Heredia Vázquez
A01650574

Samuel Kareem Cueto González
A01656120

Carlos Andrés Conde Besil
A01650549

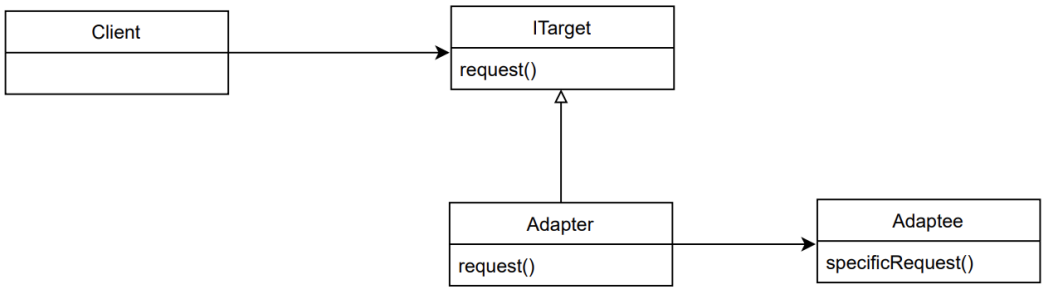
Javier Arturo Flores Zavala
A01651678

José Carlos Acosta García
A01650306

Selección de patrones a implementar

Adapter pattern

Este patrón convierte la interfaz de una clase en otra interfaz que un cliente espera. Este patrón permite que las clases funcionen en conjunto cuando sus interfaces son incompatibles. Visualmente se ve de la siguiente manera.



KEY: UML

En este caso, el cliente desea llamar a *specificRequest* pero usando la firma *request*. Es por eso que se utiliza un adaptador entre ellas. El adaptador en este caso es la implementación concreta de ITarget.

Este patrón se utiliza para alcanzar cierto nivel de interoperabilidad. Es decir, cuando deseas cierto comportamiento, pero no puedes tenerlo utilizando la interfaz que tienes. Un ejemplo de su uso puede ser que cierta librería va a cambiar el orden de los parámetros que recibe una función. Si no deseamos cambiar cada implementación, podemos recurrir a este patrón para crear un adaptador y utilizarlo.

Su implementación en código sería de la siguiente manera.

Cliente

```
ITarget target = new Adapter(new Adaptee());
target.request();
```

Interface

```
interface ITarget {
    void request();
}
```

Adapter

```
class Adapter: ITarget {

    Adaptee adaptee;

    public Adapter(Adaptee a) {
        this.adaptee = a;
    }

    public void request(){
        this.adaptee.specificRequest()
    }
}
```

Adaptee

```
class Adaptee {
    public void specificRequest(){

    }
}
```

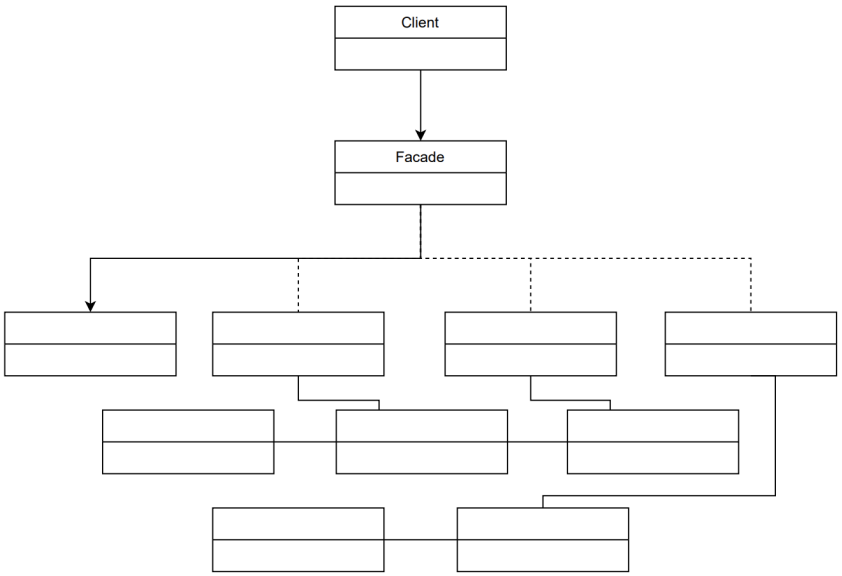
La razón por la que se desea implementar este patrón en el desarrollo del proyecto es que se hará uso de interfaces para la funcionalidad principal del mismo, pero también de librerías, APIS y componentes externos que nos ayudarán a complementarlo. Específicamente, se requiere el uso de conversión de divisas a la moneda local. Cabe mencionar que se utilizarán divisas americanas, europeas, asiáticas e incluso criptomonedas, lo que implica que se requiere acceso a APIS para cada una de las categorías mencionadas previamente. De esta manera, el usuario final no se debe preocupar en manera de entrada de datos qué tipo de moneda se envía a la cuenta destino y el adapter asiste para realizar esta conversión (por medio de API ‘s) sin que sea necesario conocer el tipo de valor porque en sí desconoce la cuenta origen.

Debido a que cada una de estas proviene de desarrolladores y fuentes de información distintas, los métodos y en general el uso es distinto. Por lo que se desea ocultar toda la complejidad de la consulta del precio en tiempo real. Es decir, se utilizará una interfaz que contiene un método genérico para consultar el precio, que a su vez estará conectado a un adaptador que desarrollaremos que llamará a los métodos específicos que correspondan dentro de cada API externa que utilicemos en el desarrollo del proyecto.

Facade Pattern

Este patrón provee una interfaz unificada a un conjunto de interfaces de un subsistema. En otras palabras, provee una interfaz de más alto nivel que hace que el subsistema sea más fácil de usar.

Su implementación consiste en que existe un cliente que desea realizar una función significativa, pero para poder hacerlo, requiere de la interacción compleja de diversos objetos. De forma gráfica se ve como en la siguiente figura.



Key: UML

Existen razones significativas por las cuales la implementación de este patrón es de utilidad en el desarrollo del proyecto. Es importante subrayar que el proyecto está apegado a los principios SOLID, específicamente al principio de responsabilidad única. Esto implica que contaremos con diversos módulos que se encargan de realizar una función específica. Por lo que, para realizar una transacción, se requiere utilizar la interacción entre cada uno de los módulos desarrollados.

En particular, para realizar una transacción, de forma general se requiere realizar la conversión de la divisa, la validación de fondos suficientes, la aprobación del pago y finalmente el envío del correo de confirmación. Si consideramos que cada funcionalidad es llevada a cabo por un módulo independiente, el proceso de realizar la transacción implica la interacción. Con el objetivo de facilitar este proceso, se implementará este patrón con una fachada que se encarga de manejar estas interacciones complejas.

Referencias

Okhravi, C. (2017). Adapter Pattern. Recuperado el 23 de abril de 2022 de: https://www.youtube.com/watch?v=2PKQtcJjYvc

Okhravi, C. (2017). Facade Pattern. Recuperado el 23 de abril de 2022 de: https://www.youtube.com/watch?v=K4FkHVO5iac