Taller 5

Samara Martínez Jácome – 202221057

1. Información general del proyecto: para qué sirve, cuál es la estructura general del diseño, qué grandes retos de diseño enfrenta (i.e. ¿qué es lo difícil?). Deben incluir la URL para consultar el proyecto.

   URL: https://github.com/peterm85/design-patterns.git
   Específicamente el fragmento donde se implementa el patrón: https://github.com/peterm85/design-patterns/tree/master/src/facade/examples/bank

   El proyecto presentado en este repositorio de GitHub es una recopilación de diferentes patrones de diseño ejemplificados. El repositorio contiene varias carpetas con los ejemplos de los patrones y su código respectivo en java. Este proyecto fue elaborado por varias personas, para tener una información más completa. El repositorio cumple con el propósito de exponer varios patrones de diseño sencillos para que los que accedan a este repositorio se lleven una idea clara de cómo se pueden implementar, y para qué sirven cada uno de estos patrones. En el README.md hay una breve explicación de cada patrón en el proyecto que funciona muy bien para entender cada uno de ellos.

   Uno de los grandes retos que enfrenta el proyecto es la consistencia en la implementación ya que en este caso como hay varios colaboradores puede que la información varie de autor a autor y la persona que consulte el código se confunda. Otro reto que presenta el proyecto es la implementación de nuevos patrones, para implementar un patrón se necesita tener una gran abstracción, y para este proyecto que su principal objetivo es exponer de una manera sencilla los patrones de diseño es un gran reto crear un proyecto corto que implemente el patrón de la mejor forma.

2. Información y estructura del fragmento del proyecto donde aparece el patrón. No se limite únicamente a los elementos que hacen parte del patrón: para que tenga sentido su uso, probablemente va a tener que incluir elementos cercanos que sirvan para contextualizarlo.

   El patrón que escogí es el de Facade. En el repositorio implementan el patrón Facade por medio de un sistema de banco. El sistema tiene 6 clases diferentes, 4 clases principales llamadas: AccountNumberCheck, SecurityCodeCheck, FundsCheck y WelcomeMessage. La primera clase es la encargada de la verificación del número de cuenta y la activación de la cuenta bancaria. La clase llamada SecurityCodeCheck es la que maneja la verificación del código de seguridad, la tercera clase manipula las operaciones relacionadas con los fondos de la cuenta así como también los retiros y los depósitos del dinero. Por último la clase WelcomeMessage es la encargada de mostrar el mensaje de bienvenida al sistema bancario.

Sin embargo la clase donde se encuentra la implementación del patrón es BankFacade. La función principal de esta clase es proporcionar una interfaz simplificada y unificada para realizar operaciones bancarias sin que el cliente necesite conocer los detalles internos de cómo se llevan a cabo esas operaciones o interactuar directamente con clases como lo son AccaountNumberCheck, SecurityCodeCheck, y FundsCheck. La clase BankFacade implementa 4 métodos que se mostraran a continuación:

- **withdrawCash**

```
public void withdrawCash(double cashAmount){

    if(canWithdraw(cashAmount)) {
        System.out.println("Transaction Complete\n");
    } else {
        System.out.println("Transaction Failed\n");
    }

}
```

Este método se encarga de manejar la solicitud de retiro de efectivo y llama al método canWithdraw para determinar si la operación es válida al final imprime un mensaje indicando si la transacción se completó o falló.

- **depositCash**

```
public void depositCash(double cashAmount){
    if(canDeposit(cashAmount)) {
        fundsChecker.makeDeposit(cashAmount);
        System.out.println("Transaction Complete\n");
    } else {
        System.out.println("Transaction Failed\n");
    }
}
```

Este método gestiona la solicitud de depósito de efectivo, utilizando el método canDeposit para determinar si la operación es válida, además llama al método makeDeposit de la instancia fundsChecker para realizar el depósito y por ultimo imprime un mensaje indicando si la transacción se completó o falló.

- **canWithdraw**

```
private boolean canWithdraw(double cashAmount){
    return accountChecker.isAccountActive(getAccountNumber()) &&
           codeChecker.isCodeCorrect(getSecurityCode()) &&
           fundsChecker.haveEnoughMoney(cashAmount);
}
```

El método actúa como una verificación compuesta, asegurándose de que la cuenta esté activa, el código de seguridad sea correcto y haya fondos suficientes para cubrir el monto solicitado para el retiro, llamando a los métodos de las otras clases del sistema para obtener esa información. Retorna un true si todas las verificaciones se cumplen.

- **canDeposit**

```
private boolean canDeposit(double cashAmount){
    return accountChecker.isAccountActive(getAccountNumber()) &&
            codeChecker.isCodeCorrect(getSecurityCode());
}
```

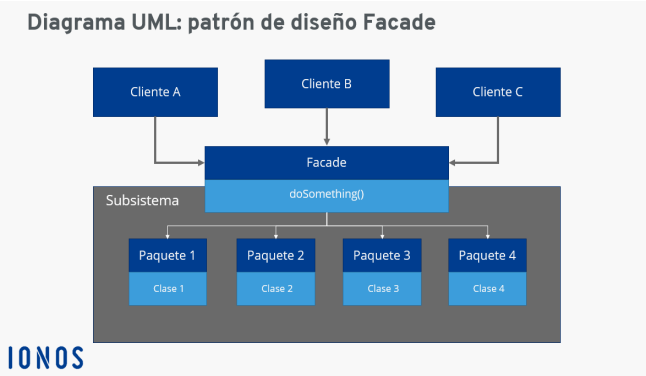
En este método se determina si es posible realizar un depósito en función de dos condiciones: que la cuenta esté activa y que el código de seguridad sea correcto. Si estas condiciones se cumple retorna un true.

En estos últimos dos métodos se encapsula la lógica interna y se coordina las verificaciones necesarias antes de realizar retiros o depósitos. Esto oculta los detalles complejos de las operaciones al cliente.

En la clase Client básicamente se inicializa el sistema bancario y se ejecuta el programa con algunos valores establecidos para poder mostrar que la interfaz simplificada de BankFacade funciona.

3. Información general sobre el patrón: qué patrón es y para qué se usa usualmente.

El patron que se presenta en el sistema es el patron Facade, este es un patrón de diseño estructural que proporciona una interfaz unificada y simplificada a un conjunto de interfaces en un subsistema. Este patrón oculta la complejidad del sistema implicito y proporciona una interfaz única para interactuar con él. Por lo tanto este patron tiene como objetivo facilitar la interfaz de un grupo de clases haciendo la interaccion con el cliente mas sencilla.



Como se evidencia en este diagrama de clases los objetos del cliente se comunican con la clase de fachada, que en este nuevo sistema es la única instancia de la que dependen directamente los clientes. Reduciendo asi la interaccion con las clases internas del subsitema. Este patron se utiliza usualmente la dependencia entre los clientes y los subsistemas debe reducirse al mínimo o cuando se planifican diferentes proyectos de software de multiples capas y los Facade ayudan a mejorar y simplificar la comunicación entre cada una de ellas.

El mejor ejemplo concreto que se le puede dar a la implementacion del Facade es el del banco, esto porque al utilizar el patron Facade se encapsulan las complejas operaciones internas, como la verificación de cuentas, la autenticación de usuarios y la gestión de fondos, proporcionando a los clientes una interfaz simple para realizar transacciones.

4. ¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?

   El Patron Facade fue una buena implementacion en el sistema bancario ya que este, como anteriormente lo explico, hace varias operaciónes complejas las cuales con las cuales el cliente no deberia interactuar. El patron de Facade esconde todos estos procesos y simplifica la interaccion con el usuario al tener una interfaz de facil comprension. De igual manera si se desea agregar una nueva clase al sistema este patron permite hacerlo facilmente debido al bajo acoplamiento entre el cliente y el subsistema, lo que permite hacer cambios en este sin afectar a los clientes. Ademas otra ventaja que encuentro de implementar este patron es que facilita la incorporación de nuevos desarrolladores al proyecto, porque la fachada proporciona una interfaz clara y documentada para las operaciones del sistema.

5. ¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

   La gran desventaja que encuentro en este patron es el alto grado de dependencia que todas las clases tienen con la clase que se utiliza como fachada. Esto es una gran desventaja porque cualquier cambio que se implemente en esta clase podria afectar bastante la relacion con las clases del subsistema. Igualmente otra desventaja que tiene este patron es la dificultada para hacer pruebas en clases individuales del subsistema porque estas pruebas siempre involucraran a la fachada completa y es mas complejo entender como se hacen las operaciones.

6. ¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?

   Otra forma que se me ocurre que podria implementarse en el sistema bancario es que en vez de implementar el patron, se podria dividir las responsabilidades del sistema bancario en módulos más pequeños y especializados. Ademas en lugar de tener una única fachada para todo el sistema, cada modulo podria tener su propia interfaz específica que exponga solo las operaciones necesarias para esa área funcional. Esto ayudaria a reducir la complejidad y a que las interfaces sean más coherentes.