Taller Integrador: Patrones de diseño

# Sistema de transacciones de cuentas.

# Objetivos

* Evaluar que patrones de diseño se deben utilizar en el desarrollo de un sistema.
* Aplicar varios patrones de diseño dentro de un mismo sistema.

# Requerimientos del sistema:

Se desea desarrollar un sistema de cajero automático que permita manejar consultas de saldo, retiros y depósitos de cuentas bancarias, para esto se le provee una clase llamada Account, la cual es la única que no debe ser modificada, pero también se proveen 3 clases más, que pueden ser modificadas a conveniencia, y sobre las que se debe aplicar los patrones de diseño elegidos.

# Desarrollar

1. Indique para cada uno de los patrones estudiados si pudiera o no servir dentro del desarrollo de este sistema. (explique)

**Creacionales**

**Factory Method**

Este patrón en este caso no es el mas conveniente, ya que se lo suele aplicar mas cuando tenemos distintos tipos de producto, pero para esta implementación solo necesitamos un tipo de producto que es cuenta.

**Abstract Factory**

Para usar este patrón, también lo mas común es que existan diferentes tipos de cuentas y que exista una clase padre abstracta que extienda a subclases que van a ser diferentes tipos de cuenta.

**Builder**

Podemos usar builder en este caso ya que nos permite que instanciamos cuentas nuevas para cada cliente, con parámetros que nos de el cliente. Además de si hubiera diferentes tipos de cuenta, el builder estaría abierto a crear nuevos tipos de cuentas.

**Prototype**

Este patrón se utiliza cuando queremos crear instancias iguales de un mismo objeto, por lo que en este caso no lo necesitaríamos porque cada cuenta es única.

**Singleton**

En este caso es útil para crear una única instancia del cajero que ya solo necesito uno que maneje las transacciones.

**Estructurales.**

**Adapter**

Adapter en este caso no nos sirve, ya que se utiliza cuando necesitamos usar un método con parámetros especiales y el objeto que queremos utilizar no es aceptado, entonces el patrón de Adapater nos permite adaptar nuestro objeto al cual necesitamos

**Composite**

Se utiliza cuando tenemos estructuras que pueden ser representadas como un árbol, en casos de que un objeto contenga a otro. Para esta practica no habría un objeto como cuenta que encierre en si a otro.

**Decorator**

Se utiliza cuando a un objeto en tiempo de ejecución queremos darle nuevos comportamientos metiéndoles en otra clase que los contengan la cual es llamada clase Wrapper.

**Facade**

Se utiliza cuando usamos una librería externa que tenga cierto nivel de complejidad, en este caso no usamos ninguna librería externa.

**De Comportamiento**

**Chain of Responsability**

Se puede utilizar al momento de realizar una transacción y haya muchos procesos de por medio como iniciar la transacción, procesos de seguridad, iniciar sesión. Para atender estas responsabilidades se crearía varios handlers y es como funciona este patrón.

**Iterator**

Como no hablamos de estructuras de datos en esta implementación no utilizaremos este patrón.

**Memento**

Cuando realizamos transacciones bancarias no se guardan estados anteriores, por lo que no necesitaremos utilizar Memento.

**Observer**

Se puede utilizar cuando se genera una nueva transacción y haya varias clases involucradas al proceso y se les notifique para que inicien un nuevo proceso según el comportamiento de la clase.

**Strategy**

Al realizar transacciones bancarias se selecciona que proceso usar por lo que podemos utilizar Strategy para que según lo que indique el usuario se utilicé la estrategia respectiva.

1. Diseñe un diagrama de clases del sistema, aplicando los patrones elegidos.

Imagen que contiene captura de pantalla

Descripción generada automáticamente