Taller Integrador: Patrones de diseño

# Sistema de Requerimientos Académicos.

# 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)
   1. Creacionales.
   2. Estructurales.
   3. De Comportamiento.
2. Diseñe un diagrama de clases del sistema, aplicando los patrones elegidos.

**CREACIONALES**

* **Abstract Factory.** - la implementación de este patrón no es necesaria debido a que el sistema no requiere que se creen diferentes productos, en este caso, diferentes cajeros con características particulares, dado a que manejamos únicamente un cajero con funciones determinadas.
* **Factory Method.-** de igual forma de abstract factory, este patrón no es útil en el sistema debido a que no es necesario la creación de algún producto extra.
* **Singleton. -** el patrón de diseño singleton es conveniente utilizar para este sistema por que se especifica que solo se tendrá un único cajero ATM por lo que solo puede existe una instancia única de ese objeto.

**ESTRUCTURALES**

* **Composite. -** este patrón no tiene mucha utilidad aquí debido a que este sistema no requiere en ningún punto una jerarquización de los componentes, además que, de las opciones existentes, no hay algunas que puedan ser agrupadas con la finalidad que estas sean tratadas de manera similar
* **Adapter .-** este patrón no es factible en el Sistema debido a que no requiere en ningún momento tratar de convertir alguna funcionalidad nueva o ya sea existente al sistema.
* **Decorator. -** este patrón nos puede ayudar en este Sistema, debido a que al tener una limitante al momento de modificar la clase Account, es posible que podamos agregarles nuevas funciones, las cuales pueden ser insertadas mediante la utilización de decoradores a lo largo de la evolución del programa.

**DE COMPORTAMIENTO**

* **Chain of Responsibility.-** este patrón si nos es útil al momento de manejar una solicitud de retirar dinero, debido a que al existir una variedad de denominaciones de billetes, este patrón nos permite solucionar dicha solicitud realizando una selección de diferentes billetes con diferentes denominaciones, pasando así de objeto a objeto; y en caso que no exista una combinación válida con respecto a la cantidad de dinero que exista en el cajero, se presentará una excepción, debido a que los objetos no pudieron manejar la solicitud.
* **Strategy. -** no se utiliza este patrón en este sistema debido a que se está manejando únicamente un tipo de problema, por lo que no es necesario realizar diferentes algoritmos que permitan resolver diferentes tipos de problemas
* **Memento. -** no es necesario debido a que, en nuestro Sistema, no es posible realizar backups de estados anteriores, pues al ser un Sistema de manejo de dinero, todos los cambios, actualizaciones y demás, se realizan en tiempo real actualizando la cantidad de dinero disponible en el sistema, sin posibilidad de volver a un estado anterior una vez realizada una acción de retiro o depósito.
* **Iterator. -** no es necesario realizar un patrón Iterator dado a que el sistema no maneja colecciones que necesiten ser recorridas para poder realizar acciones con estas.