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.**

**Factory method:** Debido a la complejidad del sistema, este no posee mas familias de objetos por lo cual la implementación de este patrón sólo permitirá aumentar la complejidad del ejercicio por lo cual este patrón no aplicaría.

**Abstract factory:** Dentro de la aplicación no se detectan familias de objetos en las que se pueda implementar dicho patrón.

**Singleton:** No es aplicable, puesto que cada vez que se realiza una transacción bancaria en el ATM este creará una nueva instancia.

* 1. **Estructurales.**

**Decorator:** Si pudiese llegar a aplicar**,** puesto que las funcionalidades de retiro y depósito pertenecen al objeto Cuenta y da la posibilidad de agregar más funcionalidades sin tener que modificarlo. Sin embargo, no se presentaría el problema de explosión de clases ya que no habría una combinación con estas funcionalidades siendo obsoleto el uso de dicho patrón.

**Composite:** No aplica para el taller, puesto que se busca la independencia entre todos los componentes, ninguna funcionalidad debe ser tratada igual.

**Adapter:** No se requiere por el momento agregar cierta funcionalidad que no es compatible con la implementación ya realizada. Por ende, no es necesario usar el patrón Adapter para acomodar una modificación no compatible.

* 1. **De Comportamiento.**

**Strategy:** Se encuentra implementado porque en la clase Manejador existen dos atributos: monto y denominación, lo que permite escoger los billetes necesarios además de la denominación. En la clase manejador podemos usar este patrón debido a que en el método retiro se pueden manejar diferentes estrategias de las formas de retiro del dinero en el cajero, validando cada una de las estrategias en un switch. No obstante, puede llegar a complicar la solución por la variedad de estrategias a implementar.

**Chain of responsibility**: Sí se puede aplicar, ya que si se requiere de una denominación específica de devolución de dinero puedo usar una lista de diferentes objetos de tipo denominación (billetes de 5, 10, 20, 50) que van a probarse para solucionar el problema de forma parcial o completa. En el caso de que este objeto no logre resolverlo seguirá al siguiente, caso contrario el problema será resuelto, es decir que si el cliente desea sacar $20 puede entregarle cuatro billetes de 5 o dos billetes de 10 o uno de 20 de acuerdo con la disponibilidad del ATM, ya que cada manejador entrega billetes de una sola denominación.

**Iterator:** No es aplicable para la clase manejador porque durante el retiro no es necesario que se iteren todos los tipos de denominaciones hasta el final, sino encontrar el tipo adecuado para entregar al cliente el dinero que desea retirar.

**Memento:** No es aplicable en este problema, ya que son transacciones de un cajero. Este patrón nos permite regresar a un estado anterior y eso no está permitido en una transacción, ya que si se retira dinero no hay forma de devolver ese dinero a ese punto.

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

