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)
   1. Creacionales.
      1. Factory Method: este patron es una generalizacion de Builder, pero para los requerimientos prosentados solo se tienen un solo tipo de “producto” (cuenta) por este motivo no es necesario el uso de este patron.
      2. Abstract Factory: Se lo podría implementar para crear cuentas, se tendría una clase abstracta AccountFactory y de esta extenderia NormalAccountFactory la cual generaria cuentas comunes y corrientes. Estaría abierto a extensiones en caso de que se requiera crear cuentas de otro tipo.
      3. Builder: Este patrón si es útil ya que con este abstraemos al cliente de la creación de las cuentas, las cuales se instanciarán inicialmente con algunos parámetros por defecto para su creación. Estaría abierto a extensiones en caso de que se requiera generar cuentas dde otro tipo.
      4. Prototype: Según los requerimientos presentados no se requiere mantener copias de instancias o respaldos.
      5. Singleton: Debido a que se va a generar un solo cajero el cual va a realizar transacciones
      6. Builder: Debido a que se van a crear 10 cuentas, entonces ya se crea un Director, con sus respectivo Builder de cuentas y el producto concreto Cuenta
   2. Estructurales.
      1. Adapter: En este proyecto no se requiere la conexión o utilización de métodos previamente implementados que requieran de un tratamiento previo para sus parámetros
      2. Composite: No se requiere el uso de este patron porque no se requiere ir recoriendo a todos los objetos compuestos en una estructura y encontrar aquel que pueda procesar una peticion, sino más bien tomar esa peticion, por ejemplo retirar dinero e ir al proceso por los diferentes Manjeadores que iran complentando el monto solicitado con su respectiva denominaci{on.
      3. Decorator: No sería necesario implementar este patrón ya que no voy a necesitar darle nuevos comportamientos a las clases existentes mediante objetos wrappers.
      4. Facade: Se va a mostrar de manera simplificada a los clientes
   3. De Comportamiento.
      1. Chain of Responsability: Se va a llevar a cabo la transacción mediante una serie de manejadores para poder realizar una transacción.
      2. Iterator: en los requerimientos no se especifica ningún comportamiento sobre iterar en alguna colección con diferentes maneras de hacerlo.
      3. Memento: No se requiere regresar de estados, por lo tanto, no se necesitaría este patrón.
      4. Observer: No se requiere un comportamiento de notificación a las cuentas sobre alguna acción por lo cual este patrón no es necesario implementarlo
      5. Strategy: No se requiere tomar una misma entrada y encontrar una solución en común, pero con diferentes estrategias. Se lo puede confundir con el comportamiento de retirar dinero, pero este es más simple y adecuado implementarlo con una cadena de responsabilidad que toma el monto faltante por completar y lo completa con lo máximo que pueda cada manejador.
2. Diseñe un diagrama de clases del sistema, aplicando los patrones elegidos.