Taller Integrador: Patrones de diseño

# Grupo 5: Ana Carrión, Carolina Zambrano, Luis Carrasco y Albin Arias

# 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.
      1. Abstract Factory. Este patrón no es útil dentro del diseño porque no existirán diferentes tipos de productos proporcionados por instancias diferentes de ATM, de hecho el sistema require solo una instancia del proveedor de las funcionalidades.
      2. Factory method. Este patrón aunque no se refiere a la familia de productos, se aplica a la diferenciación entre el comportamiento de subclases o productos en concreto; pero se repite la justificación que necesitamos un instanciamiento único para el ATM en el problema.
      3. Singleton. Este patrón limita el número de instancias posibles a uno de modo que proporciona un acceso global, lo que lo hace uno de los patrones que se pueden usar para el problema ya que en la parte principal del programa solo se necesita una instancia al ATM, porque desde el inicio de la ejecución lo correcto es administrar y gestionar la información para cada funcionalidad de un solo ATM, por ejemplo sería la cantidad de billetes de una denominación, no podría ser inicializada o cambiada desde otra instancia y eso sucede con todo dentro del sistema.
   2. Estructurales.
      1. Composite. No sirve en el proyecto que estamos diseñando. La funcionalidad de este es darle el mismo tratamiento a los objetos simples y compuestos, pero en la lógica del código no se trabaja con objetos compuestos por lo cual no es necesario.
      2. Adapter. Si sirve dentro del desarrollo del sistema. Ya que la clase Account no puede ser modificada de ninguna manera y se puede notar que solo sirve para un tipo de moneda. Por ende para que el proyecto sirva para mas de un tipo de moneda, se crea un adaptador que extiende de una interfaz con las acciones que permiten retornar información con el nuevo tipo de moneda deseada; para las condiciones particulares planteadas se involucra el currencyCode de Locale.US.
      3. Decorator. No sirve dentro del desarrollo del sistema. Para implementar este patrón se necesitaría crear una interfaz y que la clase que necesita cambiar comportamientos implemente dicha interfaz o permita que se extienda de ella, si quisiéramos darle nuevas funcionalidades a la clase Account fuera ideal pero las condiciones del planteamiento no lo permiten porque no se puede acceder al código de la misma.
   3. De Comportamiento.
      1. Chain of Responsibility. Si sirve dentro del desarrollo del sistema. Como el proyecto se trata de un ATM este patrón es el más necesario para optimizar la salida del dinero en las diferentes denominaciones; al momento de realizar un retiro de la cuenta por ejemplo se debe iniciar una cadena desde la denominación más alta hasta completar el momento para que así la cantidad de billetes o monedas que salgan del cajero sea la menor posible.
      2. Strategy. No sirve dentro del desarrollo del sistema. No existe la necesidad de encapsular un objeto del proyecto dentro de otra clase para ser utilizar algunas versiones de este, y tampoco se debe seleccionar entre muchos algoritmos para el cumplir un propósito.
      3. Memento. No sirve dentro del desarrollo porque el sistema del ATM no quiere almacenar estados anteriores de montos, cantidad de billetes por denonimación, etc. Sólo le interesa o le compete trabajar con el estado actual de cada elemento dentro del sistema.
      4. Iterator. No es útil dentro del desarrollo del sistema, principalmente porque no están requiriendo que se filtren colecciones de cuentas; no existirán colecciones de ATM porque se utilizará una sola instancia de la misma; por tanto no será necesario definir criterios o formas diferentes de recorrido para colecciones de elementos dentro del sistema.