**Patrones de diseño**

por lo genera los patrones no se escriben de a 1

3 patrones capa lógica y persistencia (todo público es un problema)

Todo patrón se basa en 3 cosas interfase

Interfase es un conjunto de operaciones que se definen sin código

public interface ix

{

void una op();

}

es una interface sin código

son las clases las que pueden implementar una interfaz (parece una herencia Y:ix) cuando una clase implementa una interface obligatoriamente debe asignar codigo a todas las op de la interface que implementa (interfaces tienen pocas operaciones)

Sirve para:

1. Para qué multiple clases que no tengan cosas en común puedan tener el mismo comportamiento.todos los datos tiene la interfaz icomparable(CompareTo)
2. Es para enmascarar objetos,brinda mayor seguridad te deja trabajar con el objeto de la clase a través de la interface pero no sabemos que tiene la clase

CompareTo siempre devuelve 3 valores 1, 0, -1 menor igual mayor Yo defino con que comparó nombres cédulas etc etc

Ver patrón: observer

cuando un patrón me da un error es abstracto al igual que la solución siempre evaluar las consecuencias de usar un patrón y su contexto

**Patron fachada**

Cuando tengo muchos servicios resuelve de juntarlos todo en un solo servicio teniendo un solo punto de acceso.

la solucion planteada es agregar una clase a mi componente hay ponemos todos los servicios y va a ser el unico punto de acceso (unica clase publica)el resto de las clases quedan pero ocultas(internal quedan)

la concecuencia es se rompen las dependecias el cliente no se entera de lo que pasa en programa

se agrega una clase mas

todo cambio que haga en logica se pone en la fachada (todo afecta a la fachada)

se agrega un nuvel de indireccion logica y persistencia si van en fachada

a la interfas grafica y entidades compartidas no se le puede agregar patrón porque el trabajo de las capas es diferente a como trabaja el patrón

Patrón fabrica (Factory)

como brindar servicios a partir de una clase de un objeto es más eficiente hacerlo a través de un objeto el problema que nos vamos a encontrar es cómo producir esos objetos para que trabajen y que salgan del componente

Problema de la fabrica:

Es un patrón que nos va a permitir crear y enviar hacia el exterior de componente objetos para que brinden servicios pero sin que se sepa su tipo concreto (Clase que creó el objeto)

El uso de interfase puede permitirnos:

Crear un objeto y devolverlo en nombre de una interface y desde afuera del componente solo se va a poder reconocer la interface pero no las clases reales que crearon el objeto (la clase de new trabajo con objetos y mas seguridad)

El patron se aplica (solucion) las clases de trabajo de mi componente donde se define nuestro servicio para cada una de las clases de trabajo vamos a tener que generar una interface que permite enmascarar objetos de dicha clase.

Crear una sola interfaz general los siguientes inconvenientes:

1. Clases de trabajo tienen que tener mismas operaciones
2. Operaciones genéricas (Código defensivo extra)

lo próximo a realizar en la solución es agregar la fabrica de objetos (asi podemos capturar los objetos (datos públicos))

una operación para cada tipo de objeto que quiera fuera del componente dichas operaciones son solamente para crear new

Consecuencias

1. Se rompen dependencias entre formulario y logica (Se desacoplan capas)
2. Trabajo con objetos (Dejo clases [Mejor que objeto brinde servicio que clase a todos los usaurios])
3. Agrego clase “Fabrica” y tantas interfaces como tipo de objetos que quiera enmascarar
4. No hay forma de manejar la cantidad de objetos (problema con la memoria del servidor)[se puede anular esta consecuencia convinando con el patron singleton]

La persistencia se muda a la fabrica

La capa persistencia brinda servicios por eso sirve puedo poner otra fábrica

Lógica brinda servicios se muda a la fábrica y trabaja con objetos

Entidades compartidas no aplica fábrica ya que no brinda servicios

La interfaz gráfica tampoco lo aplica

El tipo de acceso no existe a nivel de operación sino a nivel de método (cuando se implementa el codigo hay si se define que tipo de acceso tiene public internal …..)

la regla para cuando una clase implementa una interface dice que las implementaciones de la interface no pueden tener un tipo de acceso menor que el propio de la interface

el tipo de devolución de una operación es el tipo de dato por el que se reconoce el objeto devuelto

patrón singleton

ataca el problema de la cantidad de intancias que puede tener una clase es un patron que me define hasta cuantos quiero (patron fexible)

este patron no sirve para aplicar sobre componentes es para aplicar solamente sobre clases

en lo general una clase en un componente tenga comportamiento singleton por lo general se lo terminó aplicando al resto de las clases (no es necesario que lo aplique sobre todos los componentes)

lo unico a modificar es la propia clase y son 3 elementos que armo en la clase para programar el patron

1. Constructor privado en la clase que se va a realizar el singleton(solo la clase produce objetos) - Objetos contabilizados por la misma clase
2. Atributo de clase privado ( Almacenar las instacias de la clase) (Permite a clase crear objeto y utilizarlo ella) si hago un sigleton bien echo el atributo guarda un objeto(si quiero mantener mas cantidad mi atributo va a ser un conjunto
3. Acceso a objetos de la clase (Mediante operacion o propiedad) - Esta propiedad u operacion es de clase y publica

**Es para aplicar en clases no componentes**

Solo “get” no se setea desde afuera