TRABAJO DE CONSULTA

**NESTOR CASTRO**

**JAVIER**

UNIVERSIDAD CENTRAL

FACULTAD DE INGENIERIA

DEPARTAMENTO DE INGENIERIA DE SISTEMAS

JULIO 2015

BOGOTA D.C.

PATRONES DE DISEÑO

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos.

El concepto de patrones de diseño fue el resultado de un trabajo realizado por un grupo de 4 personas (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos como "la pandilla de los cuatro") que se publicó en 1995 en un libro titulado "Patrones de diseño: Elementos de software orientado a objetos reutilizables" en el que se esbozaban 23 patrones de diseño.

Un patrón de diseño puede considerarse como un documento que define una estructura de clases que aborda una situación particular.

Dentro del curso hemos visto los siguientes patrones los cuales a continuación describiremos en el por qué se utilizaron o no dependiendo del desarrollo del problema planteado.

**PATRON MVC**

*Problema:* Acoplamiento de la solución del problema con la visualización e intervención con el usuario.

*Solución:* Separa la visualización de la lógica del problema.

*Implementación en el proyecto:* Si, Es la base de la solución al problema mediante este patrón separaremos la parte lógica del negocio de la visual y estableceremos mediante los controladores la interacción de estos.

**PATRON VISITOR**

*Problema:*

*Solución:*

*Implementación en el proyecto*: No,

**PATRON ITERATOR**

*Problema:*

*Solución:*

*Implementación en el proyecto*: No,

**PATRON MEMENTO**

*Problema:*

*Solución:*

*Implementación en el proyecto*: No,

**PATRON CADENA DE RESPONSABILIDAD**

*Problema:*

*Solución:*

*Implementación en el proyecto*: No,

**PATRON COSTRUCTOR**

*Problema:* Cómo separar los objetos complejos

*Solución:* Una clase que construye los objetos complejos. Se sugiere crear otra clase que controle el proceso de creación, además se sugiere que el constructor implemente los métodos de construcción definidos en una interface.

*Implementación en el proyecto*: No, el proceso de creación el patrón factoría nos soluciona de manera adecuada el requerimiento planteado.

**PATRON PROTOTIPO**

*Problema:*

*Solución:*

*Implementación en el proyecto*: No,

**PATRON SINGLETON**

*Problema:* Múltiples clientes que referencian objetos de la misma clase, los cuales tienen diferentes valores en sus atributos, causando errores en la ejecución porque todas las instancias deben tener el mismo valor en sus atributos.

*Solución:* Crear una sola instancia haciendo el método constructor privado y creando un método que retorna la instancia creada/existente de la clase.

*Implementación en el proyecto*: Si, nos asegura que la clase BD solo tiene una instancia y nos provee un punto de acceso global a esa instancia.

**PATRON MONITOR**

*Problema:* Garantizar que la información es integra

*Solución:* Sincronizar el acceso a la información, asegurando que todos ven los cambios de manera inmediata y que deben esperar a que los cambios sean perdurables para hacer sus propias modificaciones.

*Implementación en el proyecto*: No, la integridad de la información la garantizamos en la BD donde cada reserva queda almacenada de acuerdo a la solicitud realizada y esta no puede volver a ser tomada.

**PATRON INMUTABLE**

*Problema:* La no modificación del contenido de un objeto

*Solución:* Indicar/asegurar que el objeto solo puede tomar valores una vez, normalmente durante su creación.

*Implementación en el proyecto*: No, No existe un acceso simultáneo a los objetos que permita el cambio constante de estos y así afecte de manera directa los objetos.

**PATRON FACTORIA**

*Problema:* Delegar la responsabilidad de crear objetos; la clase no debe tener la responsabilidad de conocer e instanciar los objetos de otras clases permitiendo aumentando la flexibilidad a desacoplar las relaciones a otras clases.

*Solución:* Delegar a creación de los objetos a otra clase que ofrece un método que devuelve la instancia requerida.

*Implementación en el proyecto*: Si, se implementa en la creación de los tipos de zonas que dispone de la universidad para poder realizar la reserva.

**PATRON FACTORIA ABSTRACTA**

*Problema:* Un marco para estandarizar el modelo arquitectónico para una gama de aplicaciones, pero permitir aplicaciones individuales para definir sus propios objetos de dominio y prevén su instanciación

*Solución:*

*Implementación en el proyecto*: No,

**PATRON FLYWEIGHT**

*Problema:* Cómo hacer para eliminar a redundancia

*Solución:* Separar las clases diferentes, la información intrínseca y la información extrínseca. La implementación intrínseca la agrupó en un objeto de peso ligero, sea un singleton o un objeto inmutable. La información extrínseca se define en otra clase que además tiene una referencia al objeto de peso ligero.

*Implementación en el proyecto*: No, no es necesario crear objetos a tan bajo nivel ya que la flexibilidad del sistema se da en que la factoría genera únicamente las zonas requeridas por la reserva, ni más ni menos.

**PATRON ADAPTER**

*Problema:* Cómo integrar una clase que soluciona el problema a un cliente, el cual además a provisto la interface que quiere que la clase cumpla.

*Solución:* Crear un adaptador que implemente la interface del cliente y solucione el problema de este haciendo uso de la clase que sabe cómo resolver el problema.

*Implementación en el proyecto*: No, la aplicación a desarrollar no contiene clases incompatibles entre ellas que nos lleven a utilizar este patrón.

**PATRON FACADE**

*Problema:* Cómo uso los servicios de un sistema

*Solución:* Crear una interfaz que indique cómo se hace uso de los subsistemas.

*Implementación en el proyecto*: Si, es la puerta de entrada a todo nuestro sistema enfocando en un solo punto el acceso a este.

**PATRON DECORATOR**

*Problema:* Cómo evitar la utilización de las subclases, generacionales o extensiones.

*Solución*: Crear una nueva clase que tenga la nueva funcionalidad y tenga como atributo a la clase de la cual desea extender o agregar las nuevas funcionalidades.

*Implementación en el proyecto*: No, las funcionalidades planteadas inicialmente en el aplicativo son gestionadas por la clase reserva la cual recibe de la factoría la zona solicitada.

**PATRON COMPOSITE:**

*Problema*: Cómo agrupar los objetos individuales a partir de objetos más sencillos para crear objetos complejos

*Solución:* Agrupar en componentes funcionales.

*Implementación en el proyecto*: No, los objetos más sencillos no son necesarios plasmar ya que estos están contenidos dentro del patrón factoría que nos ayuda a generarlos sin requerir intermediación alguna.