Modelado y Programación
Practica 3

27 de Noviembre de 2020

Integrantes:

- Gutiérrez Sánchez Claudia Itzel. No. de Cuenta: 112002433
- Montiel Manriquez Ricardo. No. de Cuenta: 314332662

Parte Teórica:

1.- Menciona los principios de diseño esenciales de los patrones Factory, Abstract Factory y Builder. Menciona una desventaja de cada patrón.

Factory:

Es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Una de las desventajas que puede presentar este patrón es que puede requerir crear una nueva clase simplemente para cambiar la clase de Producto.

Abstract Factory:

Es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas.

Una desventaja es que si queremos agregar un nuevo producto debemos implementar su interfaz y todos sus métodos

Builder:

Es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

Una de las desventajas principales es la necesidad de mantener la duplicidad de atributos que deben estar en la clase destino y en el builder.

Parte Practica:

Ejecución de la Practica:

Estando dentro de la carpeta ‘ src ’ abriremos la terminal y se ejecutara el comando ‘ javac Main.java ’ para compilar el programa. Después ejecutaremos el comando ‘ java Main ’ y con esto se mostrarla la simulación pedida.

Justificación del Patrón:

Después de considerar utilizar Abstract Factory, la descartamos como opción porque a pesar de que realizaba lo que necesitaba, Builder era una opción que podía resolver el problema también y sin demasiadas clases.

Aunque la ventaja que tiene Abstract Factory es que todo se realiza en una sola llamada, es decir, no se construye paso a paso, al irnos por esa ruta terminábamos con más clases y era más fácil confundirnos o complicarnos a la hora de implementar el patrón en java. Builder resultó ser una forma más concreta para lograr resolver lo que se nos pide, agregando componentes al auto paso a paso dependiendo de los componentes seleccionados por el cliente todo esto a través del builder y de su método crearAuto()

Originalmente habíamos implementado Abstract Factory, pero no estábamos completamente convencidos con la cantidad de clases, ya que creíamos que era posible resolver el problema sin tener tantas clases, siendo algunas de estas muy pequeñas (llegando al punto en el que teníamos clases que contenían solo uno o dos métodos).

Después de deliberar un par de horas e investigar más a fondo los patrones que podíamos ocupar, nos pareció que Builder era más amigable a la hora de implementarse. A diferencia de Abstract Factory y Factory que funcionan con una clase general (el tipo de componente), una clase para cada componente concreto de cada tipo y una interfaz general para todos los componentes, sin la necesidad de crear una fabrica para construir cada uno de ellos. Así que la cantidad de clases que teníamos originalmente disminuyo.

Consideraciones:

Tomar en cuenta que a la hora de ejecutar el programa si en algún punto el cliente se queda sin presupuesto a la hora de realizar su pedido el programa se acaba. Nosotros lo estamos tomando como que no se le está cobrando aún porque el auto no está terminado, entonces creemos que sin problema podemos sacarlo del programa diciéndole que tiene fondos insuficientes para lo que quiere.

Al iniciar la ejecución del programa debemos tomar en cuenta que no aceptamos un presupuesto menor a 7600 porque el auto mas barato a construir en esta simulación es de ese costo incluyendo a los tres autos dados por defecto, entonces no tiene sentido aceptar un presupuesto menor a ese porque nunca podrá construir un coche.