Universdad Nacional Autónoma de México

Practica 4

Modelado y programación

Equipo:
NoSeQueHacerConMiVidaException

Integrantes:

- Martínez Cano Ricardo Iván 320283284
- Méndez Razo Carlos Geovanni 315057249
- Vidal Aguilar Diego Jesus 319297591

1. Notas de la practica

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

Patron Factory:

- Es un patrón de tipo creacional que proporciona una interfaz (creator) para crear objetos en una superclase por medio de un método fabrica en vez de utilizar el operador new. Este método fabrica es implementado por todas las subclases la interfaz (concret creator) y es donde se define que clase (producto concreto) se va instanciar.
- Este patron es usado para separar el código de cosntrucción del producto y el código que hace uso del producto, facilitando la extensión y el mantenimiento.
- Una desventaja es que el código se puede complicar a medida de que vayan incorporando más tipos de productos, ya que para usarlos requeriran de más fábricas con la misma forma de instanciar las clases productos (lo que también podría traer problemas de código duplicado).

Patrón Abstract Factory:

- Proporciona una interfaz (abstract factory) que permite crear una familia de objetos relacionados (concrete products) sin conocer los detalles de su implementación. Del abstract factory se derivan subclases (concret factory) que van a devolveran los productos de un tipo en particular y que son compatibles.
- Este es utilizado cuando necesitas crear múltiples tipos de objetos que están interrelacionados de alguna manera, como diferentes componentes de un sistema que deben trabajar juntos de manera consistente.

- Tiene la misma desventaja que el patrón anterior, además de una mayor rigidez ya que la interfaz abstract factory proporciona una estructura fija para una familia de productos, por lo que si la familia de productos presenta modificaciones, habrá que modificar la interfaz y las subclases que la implementan.

Patron builder:

- Este patrón nos permite crear objetos complejos paso a paso, por medio de una interfaz (Builder) que define los pasos de la construccion del objeto. Hay subclases que implementan esta interfaz (concrete builder) que se encargan de ensamblar y construir las partes específicas del objeto (product) de forma particular. Y esta forma de construir está dirigido por una clase directora que define el orden de los pasos de la construiccion.
- Este patrón se usa para evitar clases con contructores demasiado complejos, además de separar la construcción y la representación final del objeto. Tambien permite la reutilización del código ya que la construcción del objeto se encuentra en el constructor y puede ser utilizado en varias representaciones de productos.
- Una desventaja que tiene el patrón es la complejidad general del código aumenta, ya que el patrón exige la creación de varias clases nuevas, además de que puede haber rigidez debido a la alta dependencia del cliente con la interfaz Builder.

2. Justificacion del uso de Factory.

Escogimos el patrón Factory ya que a nuestro parecer fue el que mejor se adaptaba a la problemática de la práctica, debido a la implementación de las clases fábrica para construir cada una de las partes de la nave. Estas tenían el método fabrica que podían recibir fácilmente la opción que escogiera el usuario, por ejemplo entre las distintas clase arma, e instanciar la clase arma correspondiente.

Descartamos el uso de Abstract Factory ya que no vimos la necesidad de separar los productos (los tipos de cabinas, armas, blindajes y sistemas de propulsión) en grupos o familias debido a que no había diferentes tipos de naves. Todas las naves podían ocupar los mismo productos sin problema siempre y cuando alcanzara el presupuesto.

Descartamos también builder ya que lo asociamos más para casos en el que se usen clases con constructores grandes o con muchos atributos opcionales y para esta práctica no vimos el uso de este tipo de constructores. Ademas de lo mencionado en la clase sobre la gran cantidad de informacion que debia poseer el cliente para poder sacarle provecho al patron, asi como la complejidad que adquiere el codigo.

3. Notas sobre la práctica

Algunos puntos que considero deberian ser aclarados sobre la practica, es que se usaron distintos factory que fueran llamados en nuestra clase NaveFactory debido a que se buscaba respetar que si en algun momento un nuevo tipo de parte se crea, esta pueda ser añadida al codigo sin afectar a gran parte de este y con una mayor facilidad en la lectura

Estas distintas fabricas, fueron hechas por cada componente de nuestras naves y las interfaces que instancian tambien son una por cada tipo de parte, esto buscando respetar que si en algun momento alguna parte deja de afectar una estadistica, pueda ser modificado en su interfaz sin afectar a las otras partes de nuestra nave

La practica se compila desde el src debido a la paqueteria Factory, dado que no estan importadas todas las clases en el main, es recomendable compilarlas desde sus correspondientes carpetas

En la clase menu se guardaron algunos metodos privados para facilitar la realizacion de este y al igual reducir la carga de metodos y codigo en general en nuestra clase main