UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ciencias

Equipo NullPointerException:
Adrián Aguilera Moreno - 421005200
Diego Angel Rosas Franco - 318165330
Marco Antonio Rivera Silva - 318183583

Modelado y programación

Proyecto 01

Instrucciones de instalación, compilación y ejecución.

Se dará por hecho que el usuario sabe moverse en terminal.

Requerimientos previos:

- Se debe contar con una versión de Java 8 o superior en su computadora. De preferencia la versión más reciente.

Ejecución del proyecto:

- Si está leyendo esto significa que desempaquetó con éxito el proyecto.
- Abra su terminal y diríjase a la ruta donde desempaquetó el proyecto.
- Una vez estando en la ruta

  `Proyecto01_NullPointerException`,

  diríjase a

  `Proyecto01_NullPointerException/src/fciencias/modelado/`
- Ejecute: "`javac CheemsMart.java`", esto generará los .class del proyecto.
- Ejecute: "`java CheemsMart`", esto ejecutará el proyecto mostrándole el menú solicitado para la practica.

Justificación de uso de patrones:

Strategy

Utilizamos Strategy en en la parte de cambiar el idioma debido a que lo resuelve de una forma conveniente ya que teníamos pensado utilizar State, sin embargo notamos que a pesar que en ambos tendríamos que modificar el idioma en tiempo de ejecución había una gran desventaja al usar State, por ejemplo limitar los estados con múltiples condiciones para evitar que se tuviera más de un estado a la vez, en cambio Strategy podíamos controlar que en efecto solo tuviera un idioma de una manera mucho más sencilla y efectiva, además de que sería sumamente sencillo agregar nuevos idiomas en Strategy que en State.

Decorator

Implementamos el patrón de diseño Decorator en la parte de descuentos, pues los productos pueden aplicar a uno o más descuentos, incluso del mismo tipo (al menos no hay una restricción para esto) de manera conveniente en nuestra implementación, donde el `Producto` original sin modificaciones en precios es el centro del decorador.

Otros patrones que se pudieron usar:

Singleton

Nos percatamos que podíamos haber usado Singleton en la clase Servidor, pues de esa forma tendríamos un servidor centralizado en el que tendríamos a todos nuestros usarios y el catálogo de la tienda. Sin embargo, vimos que esto podría causar un problema de rigidez, pues a futuro es posible que la tienda planeé tener servidores a lo largo del mundo para mejorar la velocidad de respuesta acorde a la región en la que se encuentre el usuario (teoricamente hablando), y en cada servidor, regional por ejemplo, tener un catálogo distinto a futuro, ya que actualmente solo se tiene un catálogo en el mismo idioma para todas las regiones. Debido a esto es que Servidor cuenta con un identificador único, para asi poder diferenciarlo de los demás servidores futuros.

Diagrama UML: