Diseño de Software

escuela superior politecnica del litoral | primer término 2018-2019

Proyecto de Diseño de Software

implementación de linea blanca S. A



# Integrantes

* Jacome Ginger
* Morán Emilio
* Ortiz Eduardo
* Vallejo Steffany

# Título del Proyecto

Implementación de Línea Blanca S.A.

# Objetivo del Proyecto

Implementar el sistema de Línea Blanca S.A utilizando patrones y refactoring en el código fuente además de las comprobaciones por medio de pruebas unitarias.

# Descripción

Línea Blanca S.A. es un negocio de ventas de artículos como refrigeradoras y cocinas de inducción. Además, cuenta con una clientela y sus trabajadores, como espacios de trabajo. Para empezar, se identifica los patrones de diseño para implementar este sistema agrupados en paquetes separados según MVC.

Después se verifica cada patrón, se realiza refactoring y así eliminar malos olores, por último, se hacen pruebas unitarias para verificar el funcionamiento de diferentes códigos de nuestro programa final.

# Patrones de diseño

1. **Factory Method**

Se utilizo este patrón para la creación de objetos dentro del sistema cuando el administrador los ingresa. Se utiliza este patrón creacional debido que nos permite crear diferentes objetos, pero relacionados, en este caso son artículos; otro punto es porque el administrador ingresa un artículo, por lo cual no se muestra la lógica de la creación.

1. **Strategy**

Este patrón de comportamiento, nos permite encapsular la forma de pago del cliente ya que no sabemos cómo el cliente pagará y tampoco nos interesa dentro el sistema por eso se utiliza este patrón con el fin de que el cliente en el sistema invoque este método.

1. **Decorator**

Se implementó este patrón, ya que, al momento de emitir una petición de editar venta, el vendedor debería de ser capaz de modificar los datos de una venta, por lo que es necesario añadir dinámicamente una nueva funcionalidad al vendedor, el cual es un objeto ya instanciado.

1. **Observer**

Este patrón es usado al momento de que el vendedor requiere permisos de edición de venta, ya que debe notificar al administrados. Por lo que el administrador en esa notificación recibida debe confirmar el permiso o no. El patrón nos ayuda para notificar a los observadores, en este caso el administrador.

1. **Chain of Responsability**

Este patrón nos permite implementar el modo de atención por parte de los vendedores hacia un cliente que acaba de llegar, delegando la atención del cliente a un vendedor que no se encuentre ocupado. En la cual, nuestro único handler es el vendedor o los vendedores logeados en el sistema, quienes atenderán el requerimiento cuando estén libres.

# Refactoring

A continuación, se describen los malos olores encontrados y técnicas de refactorización utilizadas en nuestro proyecto para eliminarlos.

* Método largo

En la clase ConsultasDB, el método consultarArticulo() tenía aproximadamente 30 líneas por lo tanto se utilizó la técnica de refactorización extraer métodos, separando cada sección de consulta en nuevos métodos que serán llamados desde el método consultarArtciulo().

Asimismo en la clase IngresoDB, en el método ingresarPago(), se utilizó extracción de métodos: ingresarPagoEfectivo() e ingresarPagoCredito().

* Lista larga de parámetros

El método agregarCliente() en la clase IngresosDB, recibía todos los atributos de un cliente, conteniendo más de 4 parámetros en el método. Por lo tanto, se aplicó la técnica de conservar todo el objeto. Y así se sustituyó todos los parámetros por un objeto cliente.

* Campos temporales

En la clase AdministradorView, se utilizaba una variable por cada creación de articulo, por lo tanto, se elimino esta variable pasando el valor directamente sin crear estos malos olores.

* Código duplicado

En la clase CargasDB, se duplicada código en varios métodos, por lo tanto se utilizó la técnica de extraer métodos, un nuevo método con el nombre de setearDatos() y en la clase ReportesDB se extrajo el método guardarDatosRep().

* Comentarios

Algunos comentarios innecesarios en algunas clases del paquete model, por lo tanto, se aplicó la técnica renombrar método.

* Código muerto

Se eliminar métodos en la clase artículo que no se utilizaban, y en sus hijas, así también en la clase vendedor y administrador.

# PRUEBAS UNITARIAS

Se realizaron pruebas unitarias debido a que facilitan los cambios y favorecen la integración, permitiendo modificar fragmentos de código sin la necesidad de afectar al mismo, posibilitando la corrección de bugs por lo cual se implementaron las siguientes pruebas:

Clase ActualizacionesDB:

* void testDatosActualizarVenta ()->assertTrue

Test para comprobar que las actualizaciones en la base de datos de un dato que no se encuentra no existen

* void testMostrarMensaje ()

Test para comprobar los mensajes emergentes al usuario.

Clase CargasDB

* testLogin ()->assertEquals

Test de comprobación de inicio de sesión.

Clase ConexionDBM

* void testEstablecerConexion ()->assertNotNull

Test para comprobar la conexión entre la base de datos y el sistema implementado.

Clase ArticuloFactory

* void testCrearArticulo ()->assertNull

Test para comprobar que la fábrica puede generar artículos.

Cotizacion

* void testCalcularTotal ()->assertFalse

Test para comprobar que el valor de la cotización es correcto.

Clase Credito

* void testGetModo ()->assertEquals

Test para comprobar que a partir de 1 mes de crédito el modo de pago será diferido.

Clase FormaPago

* void testPagar ()->assertNotNull

Test para comprobar que, a partir de un modo asignado de pago, se generara una forma de pago acorde a lo indicado.

Clase Gerente

* void testGenerarReporteVendedor ()->assertSame

Test para comprobar los reportes generados por el vendedor.

Clase Vendedor

* void testConsultarClienteEquals ()->assertNotSame

Test para verificar que los datos de dos clientes no son los mismos a pesar de que provengan de la misma clase instanciada.

* void testConsultarArticulo ()->assertNull

Test para comprobar la existencia de un artículo a partir de la solicitud de un vendedor.

* void testConsultarClienteEmpty ()->assertTrue

Test para verificar que los resultados de búsqueda de un cliente que no está en la base de datos, serán vacíos.

Clase Venta

* void testCalcularValorTotalEquals ()->assertEquals

Test para comprobar que el valor total de la venta es correcto.