**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**PROYECTO DE PRIMER PARCIAL**

**IMPLEMENTACIÓN DEL SISTEMA PARA LA EMPRESA LÍNEA BLANCA S.A.**

**INTEGRANTES**

· **Burgos Pérez, Carolina Alexandra**

· **Lecaro Lanchang, Manuel Nicolás**

· **Montenegro Pozo, Daniela Carolina**

· **Moya Larrea, Luis Ángel**

1. **Descripción de proyecto**

Se plantea el diseño de un sistema con la arquitectura MVC para facilitar la interacción existente con los usuarios por medio de una interfaz gráfica que estará relacionada con un modelo que represente las entidades involucradas en las actividades de la empresa.

Para la exportación e importación de la información diaria ya sean registros de transacciones, empleados, clientes y el inventario, se conectará el sistema a una base de datos relacional y así asegurar la permanencia de los registros durante el tiempo de vida del sistema.

1. **Patrones de diseño**

* Strategy Method

Debido a que un cliente puede seleccionar el método de pago a utilizar, ya sea tarjeta de crédito o efectivo, se decidió usar este método para que una interfaz denominada *Pago* sea implementada por los objetos *PagoTarjetaCredito* y *PagoEfectivo* y así puedan sobreescribir el método *metodoPago()*. Es decir, con este patrón se piensa encapsular el algoritmo de pago dentro de la interfaz, para luego ser sobreescrita según las necesidades de los objetos que implemente dicha interfaz.

* Chain of responsibility

Se decidió implementar este patrón gracias a que un cliente debe ser asignado a un vendedor disponible para ser atendido, por ende se necesita buscar entre todos ellos hasta que uno se encuentre libre para poder al siguiente cliente en espera. Es decir, este patrón permite asignar una responsabilidad a un conjunto de objetos vendedores.

* Decorator

Se implementó este patrón para los permisos que los distintos usuario usuarios poseerán, esto debido a que cada uno podrá realizar diferentes acciones sobre los datos como modificar, crear, leer y eliminar.

1. **Refactoring**

Para eliminar los distintos malos olores (code smells) que el proyecto presentaba, se decidió aplicar ciertas técnicas de refactorización listadas a continuación:

* Primitive Obsession

La clase entidad posee un atributo denominado *teléfono* el cual al principio era de tipo *String* pero eso es un code smellllamado Primitive Obsession ya que usamos un tipo de dato atómico para algo que no lo es por lo que se aplicó la refactorización aplicando Extraer Clase y asi *teléfono* es una clase.

* Dead Code

Se encontró que el proyecto contenían ciertas clases con código muerto, *ControlAdministrador, ControlVendedor* y *ControlGerente*, por lo cual se decidió eliminarlas para evitar malos olores en el código.

* Duplicated code

En todos los controladores de las vistas, al cambiar de escenario, se usaba el mismo código, por lo tanto, ese método debió ser extraído y puesto en una nueva clase, que se dedicaría solamente al cambio de escenarios. La clase Escenario se encarga de abrir ventanas, notificaciones en snackbar, alertas y cambiar pantallas a partir de MouseEvent, es usada por todos los controladores, sea por invocación directa en la clase o por invocación a partir de la superclase del controlador.

1. **Pruebas Unitarias**

En el paquete *Test Packages* se encuentran varias clases que examinan ciertas partes de código en otras clases para ver si estos presentan fallos. A continuación se listan las clases del paquete previamente mencionado con sus respectivos métodos para realizar las pruebas unitarias.

**Clase:** ConexionesDataBaseTest

* public void testGetConn()

Verifica si el método *getConn()* de la clase *ConexionesDataBase* retorna un objeto,de tipo *Connection*, nulo mediante el uso de la prueba unitaria *assertNull()*.

* public void testConect()

Verifica si el método *getConn()* de la clase *ConexionesDataBase* retorna un objeto no nulo mediante el uso de la prueba unitaria *assertNotNull()*, es decir que la conexión con la base de datos sea válida.

* public void testCloseConnection()

Verifica si el método closeConection() de la clase ConexionesDataBase retorna un objeto no nulo mediante el uso de la prueba unitaria asserNotNull(), es decir que la conexión exista pero a su vez esta ya esta cerrada.

* public void testReturnView()

Verifica si el metodo retrurnView() de la clase FXMLBusquedaGenericaController retorna correctamente un tipo de permiso (string), para ello se utiliza la prueba unitaria assertEquals(). Si ambos son iguales entonces el tipo es correcto.

* public void testGetConnection()

Verifica si el metodo getConnection() de la clase FXMLLoginController retorna un objeto,de tipo *Connection*, nulo mediante el uso de la prueba unitaria *assertNull()*.

* public void testRequestUser()

Verifica si el método requesUser() de la clase FXMLLoginController retorna correctamente un null al enviarle parametros vacíos debido a que no existen dichos registros en mi base. Se usó la prueba unitaria asserEquals().

* public void testGetEmpleado()

Verifica si el método getEmpleado() de la clase testGetEmpleado retorna un empleado null utilizando la prueba unitaria asserNull().