MANUAL TÉCNICO PRACTICA 2 202003654

1. DESCIPCION GENERAL DE LA SOLUCION

A. Para la solución del sistema de administración de ordenes se utilizó java para una aplicación de escritorio con JAVA FX para un diseño de interfaces más limpio y abierto a mejores diseños.

B. Debido a conflictos de compilación de los .jar se utilizaron librerías especiales para el reconocimiento de los paquetes debido a conflictos de Maven.

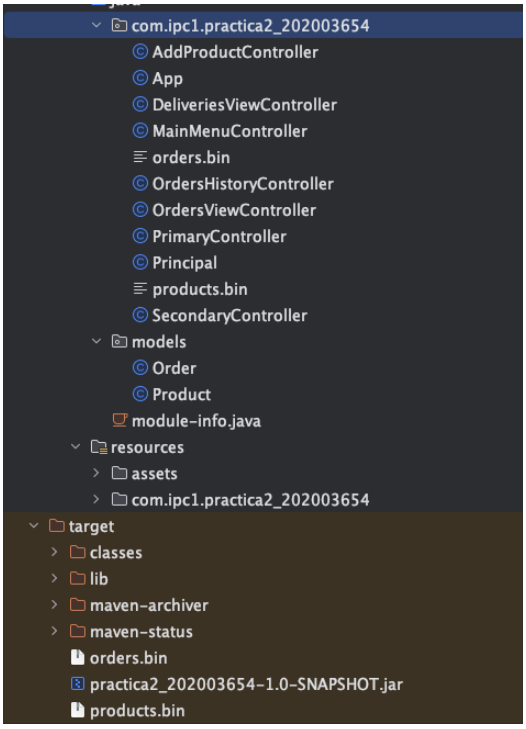
2. REQUERIMIENTOS DEL SISTEMA

A. Re recomienda el uso del openjdk 11

B. Un sistema con arquitectura x86 para mejor funcionamiento con netbeans 8gb de ram. Para arm se recomienda utilizar las versiones de azul para una compilación nativa en sistemas arm de apple.

C. Para edición de interfaces se recomienda el uso de scene builder.

3. ESTRUCTURA DEL PROYECTO



A. La estructura del proyecto se encuentra en los paquetes

i. com.ipc1.practica2_20203654

1. *Aquí encontrará los controladores principales para las funciones de las vistas, así como la clase App, Principal y los binarios que almacenará la información de las ordenes y productos*

ii. Models

1. *Aquí están los modelos para las ordenes ylos productos*

iii. Resourses

1. *Assets*
    a. *Encuentra todos los iconos utilizados en las vistas*
2. *com.ipc1.practica2_20203654*
    a. *Encuentra los archivos FXML para las vistas*

4. CLASE PRINCIPAL

A. En esta clase se importa la creada clase main que invoca la clase App la cual contiene lo necesario para el funcionamiento de la aplicación

5. CLASE APP

A. Variables globales

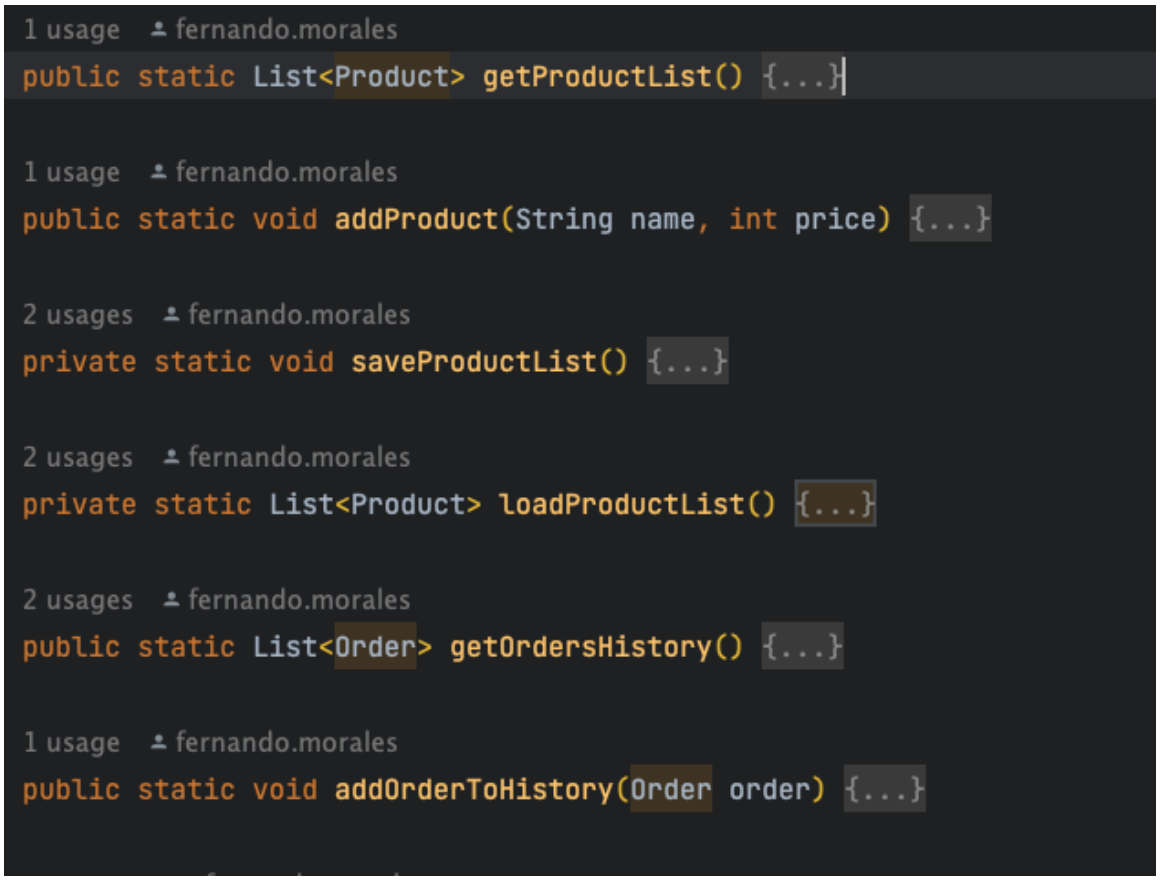
```
public class App extends Application {
    11 usages
    private static Scene scene;
    12 usages
    private static List<Product> productList;
    10 usages
    private static List<Order> ordersHistory;
    8 usages
    private static List<Order> currentDeliveryOrders;
    2 usages
    private static final String PRODUCT_FILE = "products.bin";
    2 usages
    private static final String ORDERS_FILE = "orders.bin";
```

i. Scene hace referencia al la vista actual.
ii. productsList son el listado de productos.
iii. ordersHistory almacena el historial de ordenes.
iv. currentDeliveryOrders contiene las ordenes que se encuetran listas para enviar o en proceso de envio
v. PRODUCT_FILE referencia al archivo binario que guarda los productos
vi. ORDERS_FILE referencia al archivo binario que guarda las ordenes

```
    fernando.morales
    @Override
    public void start(Stage stage) throws IOException {...}

    10 usages  fernando.morales
    static void setRoot(String fxml) throws IOException {...}

    2 usages  fernando.morales
    private static Parent loadFXML(String fxml) throws IOException {...}

    1 usage  fernando.morales
```

B. Metodos de inicio

i. Start iniciará la aplicación y cargará la vista principal
ii. setRoot cambiará la vista a mostrar
iii. loadFXML buscará la vista que se desea mostrar



C. Metodos de productos y ordenes

i. getProductsList
   1. *permite el acceso al listado de productos*

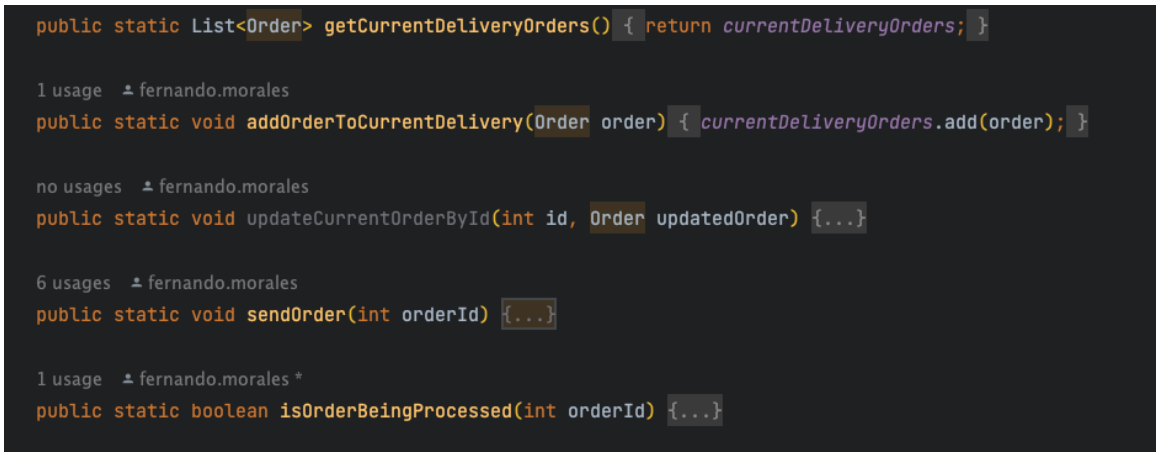
ii. addProduct
   1. *agregará un nuevo producto a la lista*

iii. saveProductList
   1. *guardará en el archivo binario el listado de productos*

iv. loadProductList
   1. *cargará el listado de productos provenientes del binario*

v. getOrdersHistory
   1. *retornará el listado de ordenes*

vi. addOrderToHistory
   1. *agregará una nueva orden al listado de ordenes*



D. Métodos para el envío de las ordenes

i. getCurrentDeliveryOrders
   1. *retornará las órdenes listas para enviar*

ii. addOrderToCurrentDelivery
   1. *agregará una nueva orden al listado de ordenes listas para enviar*

iii. sendOrder
   1. *Iniciará un hilo con el tiempo que tardará la orden en ser enviará y tomará el tiempo con base en la distancia a enviar*
   2. *Al enviar la orden la removerá del listado de ordenes por enviar*
   3. *Actualizará la orden en el historial*
   4. *Guardará los cambios en el archivo binario de ordenes*

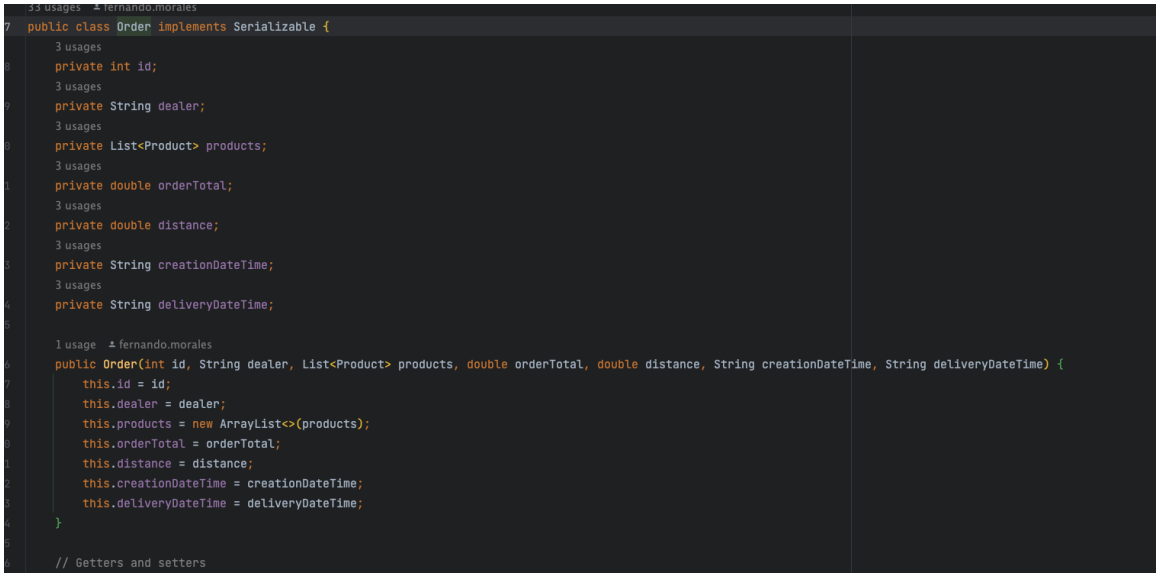
```java
30 usages  fernando.morales
public class Product implements Serializable {
    3 usages
    private int id;
    3 usages
    private String name;
    3 usages
    private double price;

    5 usages  fernando.morales
    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    fernando.morales
    public int getId() { return id; }

    fernando.morales
    public void setId(int id) { this.id = id; }

    1 usage  fernando.morales
    public String getName() { return name; }

    no usages  fernando.morales
    public void setName(String name) { this.name = name; }

    3 usages  fernando.morales
    public double getPrice() { return price; }

    1 usage  fernando.morales
    public void setPrice(double price) { this.price = price; }
}
```

E. Modelo de productos

i. Implementa un serealizable para poder ser guardado sus datos en el archivo binario.

ii. Se conforma por los siguientes valores

1. *Entero para el id*
2. *Un Sring para el nombre*
3. *Un double para el precio*



```
33 usages   fernando.morales
public class Order implements Serializable {
    3 usages
    private int id;
    3 usages
    private String dealer;
    3 usages
    private List<Product> products;
    3 usages
    private double orderTotal;
    3 usages
    private double distance;
    3 usages
    private String creationDateTime;
    3 usages
    private String deliveryDateTime;

    1 usage   fernando.morales
    public Order(int id, String dealer, List<Product> products, double orderTotal, double distance, String creationDateTime, String deliveryDateTime) {
        this.id = id;
        this.dealer = dealer;
        this.products = new ArrayList<>(products);
        this.orderTotal = orderTotal;
        this.distance = distance;
        this.creationDateTime = creationDateTime;
        this.deliveryDateTime = deliveryDateTime;
    }

    // Getters and setters
```

F. Modelo de ordenes

i. Implementa un serializable para poder almacernarlos en el archivo de ordenes

ii. Los datos son

1. *Un entero para el id*
2. *Un String para el repartidor*
3. *Un listado de productos para los productos de la orden*
4. *Un Double para el precio total de la orden*
5. *Un Double para la distancia a recorrer*
6. *Un String para la fecha de creación*
7. *Un String para la fecha de entrega*