UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1
CATEDRÁTICO: ING. NEFTALI DE JESUS CALDERON MENDEZ
TUTOR ACADÉMICO: ERWIN FERNANDO VASQUEZ PEÑATE

MANUAL TÉCNICO

PABLO DANIEL ALVARADO RODRIGUEZ
CARNÉ: 202130534
SECCIÓN: A

GUATEMALA, 18 DE DICIEMBRE DEL 2,024

ÍNDICE

INTRODUCCIÓN

Este Manual busca explicar al programador las funciones y métodos más importantes que se utilizaron para la creación del sistema y el funcionamiento de estos.

OBJETIVOS

1. **GENERAL**
   - 1.1. Dar a conocer el funcionamiento del sistema.
2. **ESPECÍFICOS**
   - 2.1. Se mostrará la forma en que funcionan los procedimientos y funciones.
   - 2.2. Mostrar la forma en que se integran las diferentes partes del sistema.

ALCANCES DEL SISTEMA

Este manual está pensado para conocer el funcionamiento interno del sistema de inventario y ventas Cobra Kai Dojo, en este se describirán los métodos más relevantes para el funcionamiento del programa.

Se explicará la forma de validar las credenciales en el login, la creación del módulo de administrador, el funcionamiento de la creación de facturas, la forma en que se ordenan los datos para los reportes así como la lectura y escritura de archivos para la persistencia de datos. Todo esto desde el enfoque técnico.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
    - Para dar continuidad a este proyecto se recomienda utilizar una PC o laptop de al menos 4GB de RAM y tener al menos 10GB libres en el sistema para la instalación del IDE de preferencia.
- **REQUISITOS DE SOFTWARE**
    - Dado que JAVA es multiplataforma puede ejecutarse tanto en entorno Linux, Windows y Mac, el sistema operativo está a elección del desarrollador. Sin embargo este sistema fue creado utilizando JDK 22 y usando el gestor Maven, por tanto el Sistema que se elija debe tener instalado JDK 22 y utilizar Maven como gestor de proyectos en el IDE.

DESCRIPCIÓN DE LA SOLUCIÓN

- Para la solución de los requerimientos se utilizó el método **divide y vencerás** donde para cada requerimiento se pensó en procedimientos y funciones propias para cada requerimiento, así como funciones y procedimientos compartidos donde más de un requerimiento utiliza un procedimiento en común.

LÓGICA DEL PROGRAMA

❖ Clase Main

```
5    import javax.swing.*;
```

➢ Librerías

La librería **java.swing.*** nos permite mostrar mensajes de forma interactiva para el usuario al hacer uso de **JOptionPane**.

➢ Función Main

En esta función tratamos de obtener los datos de los archivos mediante la clase **DeserializarObjetos()**, en caso de no haber información o que ocurra un error nos aseguramos que el sistema no se cierre y mostrará un mensaje en consola indicando el tipo de error.

```
7  public class Main {  DanielRodriguezCUNOC
8      public static void main(String[] args) {  DanielRodriguezCUNOC
9          //Intentamos cargar los datos de los archivos
10         try {
11             new DeserializarObjetos();
12         } catch (Exception e) {
13             e.printStackTrace(System.err);
14             JOptionPane.showMessageDialog( parentComponent: null,  message: "Error al cargar los datos",  title: "Error"
15         }
16         //Ingresamos al sistema
17         new Login();
18     }
19 }
```

➢ **Procedimientos, métodos y Funciones utilizadas**

A continuación se dará una explicación general sobre los métodos y funciones más importantes del sistema:

- ```
  public void addProductoIndividual(String nombre, double precio,
      if (!productExists(nombre) && precio > 0 && stock ≥ 0) {
          Producto producto = new Producto(nombre, precio, stock)
          listadoProductos.add(producto);
          confirmarAccion = true;
      } else {
          confirmarAccion = false;
      }
  }
  ```

  Este procedimiento se encuentra en la clase **ControlProductosDAO** y nos permite agregar un producto a nuestra lista de productos, obtiene los parámetros necesarios para la creación de un objeto de tipo Producto, realiza las validaciones necesarias y si todo cumple con las especificaciones realiza la acción de crear un objeto de tipo Producto y luego lo agrega a **listadoProductos**.

- ```
  public void editarProducto(String nombre, double precio, int stock) {  1 usage
      for (Producto producto : listadoProductos) {
          if (producto.getNombre().equals(nombre) && precio > 0 && stock ≥ 0) {
              producto.setPrecio(precio);
              producto.setStock(stock);
              confirmarAccion = true;
              break;
          } else {
              confirmarAccion = false;
          }
      }
  }
  ```

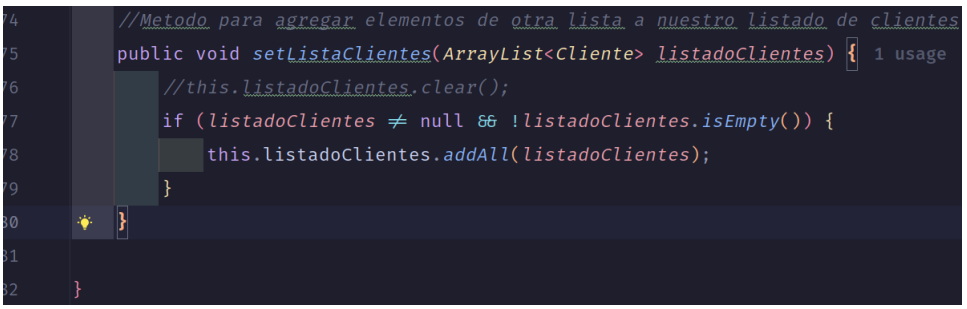
  Dentro de la clase **ControlProductosDAO** tenemos el método **editarProducto** que recibe: el nombre del producto, el precio y el stock, recorremos el listado de productos, verificamos que sea válido y se edita.

```java
//Metodo para aumentar las compras de un cliente
public void aumentarCompras(String nombreCliente) { 1 usage  DanielRodri
    for (Cliente cliente : listadoClientes) {
        if (cliente.getNombre().equals(nombreCliente)) {
            cliente.setCantidadCompras(cliente.getCantidadCompras() + 1);
        }
    }
}
```

- Dentro la clase **ControlClienteDAO** tenemos el método **aumentarCompras** que recibe una cadena luego se recorre la lista de clientes se compara el atributo de Nombre de cada cliente con la cadena recibida y si coincide aumenta el atributo cantidadCompras de ese Cliente.

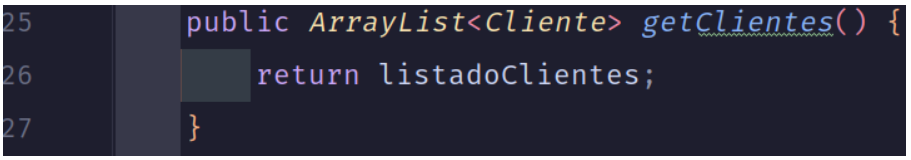
```java
public boolean clienteExists(String nombreCliente) {
    for (Cliente cliente : listadoClientes) {
        if (cliente.getNombre().equalsIgnoreCase(nombr
            return true;
        }
    }
    return false;
}
```

- Dentro de la clase **ControlClienteDAO** tenemos el método **aumentarCompras** que recibe un nombre, luego recorre la lista comparando la cadena y sin importar las mayúsculas o minúsculas. Este método es esencial para evitar clientes duplicados.

- 

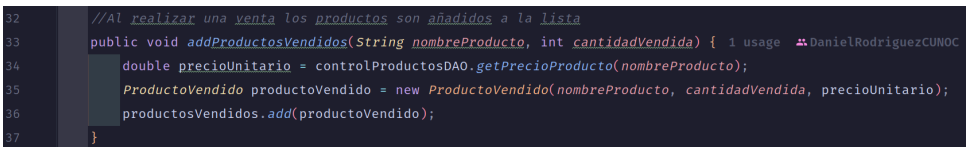
```
74      //Metodo para agregar elementos de otra lista a nuestro listado de clientes
75      public void setListaClientes(ArrayList<Cliente> listadoClientes) { 1 usage
76          //this.listadoClientes.clear();
77          if (listadoClientes != null && !listadoClientes.isEmpty()) {
78              this.listadoClientes.addAll(listadoClientes);
79          }
80      }
81
82  }
```

Dentro de la clase **ControlClienteDAO** tenemos el método **setListaClientes** que recibe una lista de clientes, esta clase es necesaria para la deserialización.

- 

```
25      public ArrayList<Cliente> getClientes() {
26          return listadoClientes;
27      }
```

Dentro de la clase **ControlClienteDAO** tenemos el método **getClientes** que devuelve una lista de clientes, esta clase es necesaria para la serialización.

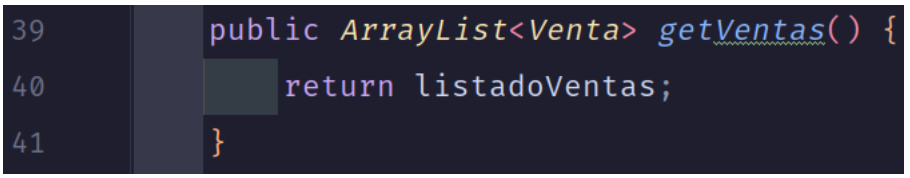
- 

```
32      //Al realizar una venta los productos son añadidos a la lista
33      public void addProductosVendidos(String nombreProducto, int cantidadVendida) { 1 usage  DanielRodriguezCUNOC
34          double precioUnitario = controlProductosDAO.getPrecioProducto(nombreProducto);
35          ProductoVendido productoVendido = new ProductoVendido(nombreProducto, cantidadVendida, precioUnitario);
36          productosVendidos.add(productoVendido);
37      }
```

Dentro de la clase **ControlVentasDAO** tenemos el método **addProductosVendidos** que recibe el nombre del producto y la cantidad que se ha vendido de este. Se obtiene el precio del producto utilizando el nombre y el método **getPrecioProducto** de la clase **ControlProductosDAO**, luego se crea un objeto de tipo Producto y se almacena en la lista productosVendidos, este método es esencial para la creación del reporte de productos más vendidos.

- 

Dentro de la clase **ControlVentasDAO** tenemos el método **setListaVentas** que recibe una lista se hacen verificaciones y se agregan los elementos al listado de clientes, luego aumentamos el tamaño del contador de ventas. Este método es esencial para la deserialización.

- 

Dentro de la clase **ControlVentasDAO** tenemos el método **setListaVentas** devuelve el listado de ventas, este método es esencial para la serialización.

```java
public void cargarClientesDesdeArchivo() { 1 usage  DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_CLIENTES/clientes.dat";
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nombreArchivo))) {
        ArrayList<Cliente> listadoClientes = (ArrayList<Cliente>) ois.readObject(); Unchecke
        controlClienteDAO.setListaClientes(listadoClientes);
        System.out.println("Clientes cargados con éxito.");
        //Archivo vacío
    } catch (EOFException ae) {
        System.out.println("No hay clientes registrados.");
        //Si no encuentra la clase Cliente o no puede leer el archivo
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al cargar clientes.");
    }
}
```

```java
public void cargarProductosDesdeArchivo() { 1 usage  DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_PRODUCTOS/productos.dat";
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nombreArchivo))) {
        ArrayList<Producto> listadoProductos = (ArrayList<Producto>) ois.readObject(); Unchec
        controlProductosDAO.setListaProductos(listadoProductos);
        System.out.println("Productos cargados con éxito.");
        //Archivo vacío
    } catch (EOFException ae) {
        System.out.println("No hay productos registrados.");
        //Si no encuentra la clase Producto o no puede leer el archivo
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al cargar productos.");
    }
}
```

```java
public void cargarVentasDesdeArchivo() { 1 usage  DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_VENTAS/ventas.dat";
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nombreArchivo))) {
        ArrayList<Venta> listadoVentas = (ArrayList<Venta>) ois.readObject(); Unchecked cast:
        controlVentasDAO.setListaVentas(listadoVentas);
        System.out.println("Ventas cargadas con éxito.");
        //Archivo vacío
    } catch (EOFException ae) {
        System.out.println("No hay ventas registradas.");
        //Si no encuentra la clase Cliente o no puede leer el archivo
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al cargar ventas.");
    }
}
}
```

Dentro de la clase **DeserializacionObjetos** tenemos tres métodos esenciales para la persistencia de datos. Cada método lee un archivo distinto en una ruta distinta. Cada método maneja 2 posibles tipos de errores: el primero es si el archivo está vacío y el segundo si la clase no se encuentra o el archivo está corrompido.

-
```java
public void guardarClientesEnArchivo() {  1 usage   DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_CLIENTES/clientes.dat";
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nombreArchivo))) {
        oos.writeObject(controlClienteDAO.getClientes());
        System.out.println("Clientes guardados con éxito.");
    } catch (IOException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al guardar clientes.");
    }
}
```

```java
public void guardarVentasEnArchivo() {  1 usage   DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_VENTAS/ventas.dat";
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nombreArchivo))) {
        oos.writeObject(controlVentasDAO.getVentas());
        System.out.println("Ventas guardados con éxito.");
    } catch (IOException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al guardar ventas.");
    }
}
```

```java
public void guardarProductosEnArchivo() {  1 usage   DanielRodriguezCUNOC
    String nombreArchivo = "DATA/ALMACENAR_PRODUCTOS/productos.dat";
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nombreArchivo))) {
        oos.writeObject(controlProductosDAO.getListadoProductos());
        System.out.println("Productos guardados con éxito.");
    } catch (IOException e) {
        e.printStackTrace(System.err);
        System.out.println("Error al guardar productos.");
    }
}
}
```

Dentro de la clase **SerializacionObjetos** tenemos tres métodos esenciales para la persistencia de datos. Cada método escribe dentro de un archivo distinto en una ruta distinta. Cada método maneja errores que pueden ocurrir durante la escritura de los archivos.