Manual Técnico
Blue Mall

Universidad de San Carlos de Guatemala

Introducción a la programación de computadoras 1

Carlos Eduardo Soto Marroquín

201902502

Segundo semestre – septiembre 2021

Contenido:

Introducción

El programa de “Blue Mall” fue un proyecto dado a los estudiantes de IPC-1, con la intención de desarrollar la lógica de orientado a objetos y que manejen lo que son las librerías swing, y awt, que son para poder manejar interfaces graficas.

Los requerimientos mínimos para poder usar dicho programa son:

Especificación Técnica:

- **Lenguajes utilizados: Java.**
- **IDE usado: NetBeans 8.2**
- **Editor de código: None**
- **Sistema operativo: Windows 10(64 bits)**
- **Interfaz gráfica: Librerías swing y awt**
- **Librerías utilizadas: Swing, awt, jFreeChart, GSON.**

Objetivos:

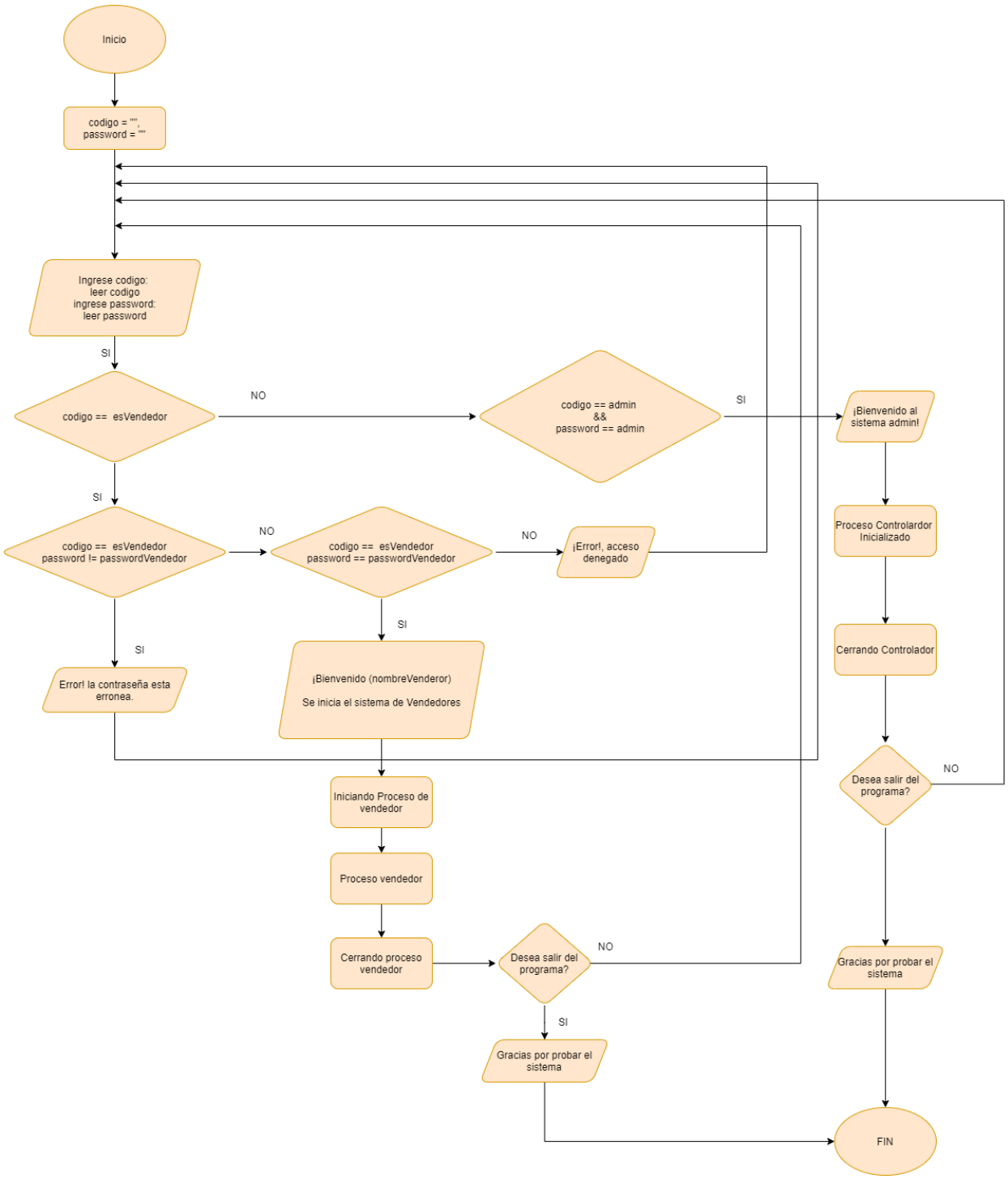
- **Uso de variables globales y locales.**
- **Uso de la memoria estática.**
- **Uso de estructuras de control y selección.**
- **Adecuado uso del paradigma orientado a objetos.**
- **Análisis y sintetización de la información.**
- **Uso de librerías externas e internas para Java.**

Alcances del proyecto

Como se mencionó de anterior manera, es que el alumno pueda aprender a utilizar y a desarrollar un programa de sistema de ventas, donde existen diferentes clases, ya que este programa está orientado a objetos, las clases que se pueden mencionar a priori son: Sucursales, Productos, Clientes, Vendedores, están son colocadas explícitamente, en mi caso cree una clase más que se llama Venta, que fue útil para poder unir los datos de la venta que realiza un vendedor a la hora de vender productos a un cliente, así mismo para poder obtener los datos de la factura.

Flujo de trabajo:

5


Inicio
codigo = ""
password = ""
Ingrese codigo:
leer codigo
ingrese password:
leer password
SI
codigo == esVendedor
NO
codigo == admin
&&
password == admin
SI
¡Bienvenido al sistema admin!
Proceso Controlardor Inicializado
Cerrando Controlador
Desea salir del programa?
NO
Gracias por probar el sistema
FIN
SI
codigo == esVendedor
password != passwordVendedor
NO
codigo == esVendedor
password == passwordVendedor
NO
¡Error!, acceso denegado
SI
Error! la contraseña esta erronea.
SI
¡Bienvenido (nombreVenderor)
Se inicia el sistema de Vendedores
Iniciando Proceso de vendedor
Proceso vendedor
Cerrando proceso vendedor
Desea salir del programa?
NO
SI
Gracias por probar el sistema

Lógica del programa

Se irá presentando cada uno de los métodos, funciones, y procedimientos que se fueron utilizando a lo largo de dicho programa para lograr su funcionamiento. Como bien se sabe, dicho proyecto se trata de una empresa que necesita un sistema de ventas, para poder vender sus productos, es porque ello que se nos entregó las partes base de con qué tipo de personas y objetos se iba a tratar.

Para esto es necesario hablar sobre las clases, ya que este programa está orientado a objetos, trataremos a todo como un objeto. Los primeros objetos que se pueden visualizar son los siguientes:

Sucursal: Podemos ver que la empresa tiene varias sucursales donde pueden distribuir sus productos, sus principales atributos son:

- Código
- Nombre
- Dirección
- Correo
- Teléfono

Estos son sus atributos, ahora bien, también se realizaron unos cuantos métodos más que son los normales para poder crear la clase de Sucursal, para esto se presenta la clase Sucursal:

```java
import java.io.Serializable;

public class Sucursal implements Serializable {

    private int codigo;
    private String nombre;
    private String direccion;
    private String correo;
    private String telefono;

    //Constructor vacio
    public Sucursal() {

    }

    //Constructor personalizado
    public Sucursal(int codigo, String nombre, String direccion, String correo, String telefono) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.direccion = direccion;
        this.correo = correo;
        this.telefono = telefono;
    }
```

Clase Sucursal, elaboración propia, 2021

Como se puede observar en la imagen de arriba están sus atributos mencionados anteriormente, además de visualizar sus constructores, uno vacío para mantener las buenas prácticas y otro ya personalizado con sus atributos respectivos, para poder inicializar un objeto.

Cliente:

Para la siguiente clase, se trabajó con lo que es la parte de cliente, el cual es que comprará los productos, para este también se tienen atributos:

- Código
- Nombre
- Nit
- Correo
- Genero

Estos son sus atributos principales, con los cuales se trabajará, ahora se muestra una imagen de la clase de cliente:

```java
import java.io.Serializable;

public class Cliente implements Serializable {

    private int codigo;
    private String nombre;
    private String nit;
    private String correo;
    private String genero;

    //Constructor vacio
    public Cliente() {

    }

    public Cliente(int codigo, String nombre, String nit, String correo, String genero) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.nit = nit;
        this.correo = correo;
        this.genero = genero;
    }
```

Clase Cliente, elaboración propia, 2021

Se puede observar comprobar que sus atributos mencionados anteriormente se encuentran ahí en la clase, y se visualiza un constructor vacío y una personalizado. El personalizado es para poder iniciar sus atributos.

Vendedores:

Dicha clase también posee sus propios atributos, los cuales son:

- Código
- Nombre
- Caja
- Ventas
- Genero

Estos atributos también se pueden observar en la siguiente imagen que contiene la clase:

```java
import java.io.Serializable;
import javax.swing.JOptionPane;

public class Vendedor implements Serializable {

    private int codigo;
    private String nombre;
    private int caja;
    private int venta;
    private String genero;
    private String password;
    private Venta ventas[];
    private static final int MAX_VENTAS = 100;
    private int contador_ventas;

    //Constructor vacio
    public Vendedor() {

    }

    public Vendedor(int codigo, String nombre, int caja, int venta, String genero, String password) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.caja = caja;
        this.venta = venta;
        this.genero = genero;
        this.password = password;

        this.ventas = new Venta[MAX_VENTAS];
        this.contador_ventas = 0;
    }
```

Clase Vendedor, elaboración propia, 2021

Cabe resaltar que está clase es importante ya que es la que nos ayudará a dar un complemento a otra clase que se mencionará más adelante, pero esta clase muestra sus métodos propios, para poder seguir trabajando con el programa.

```java
public void MostrarDatos() {
    System.out.println("\n\nLista de ventas realizadas por el vendedor: " + this.nombre);
    for (int i = 0; i < ventas.length; i++) {
        if (ventas[i] != null) {
            System.out.println(ventas[i].toString());
            System.out.println("-----------------------------------\n");
        }//sino esta en nulo
    }
}
```

Mostrar Datos, elaboración propia, 2021

Este método lo que realiza es una demostración de algunos atributos del vendedor, y además muestra un arreglo de una clase venta, esta clase es la una clase que nos ayudará a completar la información para una factura o para el registro de ventas del vendedor. Esta sirve para poder ver los datos por medio de la consola, este método esta pensando para poder ser usando a la hora de realizar una revisión del programa o mejoramiento de este.

```
public void agregarVenta(Venta venta) {
    if (contador_ventas < MAX_VENTAS) {
        this.ventas[contador_ventas++] = venta;
    } else {
        JOptionPane.showMessageDialog(null, "La cantidad de ventas ha sido superada!");
    }
}
```

Agregar Venta, elaboración propia, 2021

Dicho método nos servirá para poder agregar la venta al arreglo de ventas que posee un solo vendedor, un vendedor puede llegar a realizar 50 ventas como máximo, es por eso que cuando el vendedor pase las 50 ventas se le hará un aviso, diciendo “La cantidad de ventas ha sido superada”.

```
public void aumentarVentas() {
    this.venta = this.venta + 1;
}
```

Aumentar Ventas, elaboración propia, 2021

Dicho método nos servirá para poder aumentar el valor de las ventas del vendedor, así poder estar actualizado con la información de él(ella).

Producto:

Bien, esta es la penúltima clase, como tal posee los siguientes atributos:

- Código
- Nombre
- Descripción
- Cantidad
- Precio

Se procede a mostrar la clase de Producto:

```java
import java.io.Serializable;
import javax.swing.JOptionPane;

public class Producto implements Serializable {

    private int codigo;
    private String nombre;
    private String descripcion;
    private int cantidad;
    private double precio;

    //Constructor vacio
    public Producto() {

    }

    public Producto(int codigo, String nombre, String descripcion, int cantidad, double precio) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.cantidad = cantidad;
        this.precio = precio;
    }
```

Clase Producto, elaboración propia, 2021

Como se puede observar, tiene los mismos atributos que se mencionaron antes, además de que están ambos constructores, el vacío y el personalizado. Esta clase tiene un método aparte para poder realizar el des conteo de la cantidad de un producto.

```java
public void descontarProductos(int cantidad_a_descontar) {
    if (cantidad > 0) {
        cantidad = cantidad - cantidad_a_descontar;
    } else {
        JOptionPane.showMessageDialog(null, "No se pueden retirar "
                + "mas productos, productos agotados!");
    }
}
```

Descontar Productos, elaboración propia, 2021

Como se mencionó antes, solo descuenta cierta cantidad de productos.

Venta:

En esta parte se toca la última clase la cual, fue utilizada para poder mostrar los datos de la venta, el historial de ventas del vendedor y además para poder generar la factura.

La clase Venta, contiene los siguientes atributos:

- Vendedor
- Cliente
- Productos de la venta
- Total
- Cantidad
- Sub Total
- Fecha
- No Factura
- Contador de productos
- MAX_PRODUCTOS

Bien, para poder llevar mejor explicación es necesario, mostrar la imagen de la clase:

```java
import java.io.Serializable;
import javax.swing.JOptionPane;

public class Venta implements Serializable {

    private Vendedor vendedor;
    private Cliente cliente;
    private Producto[] productos_venta;
    private Double total;
    private int[] cantidad;
    private Double[] subTotal;
    private String fecha;
    private int noFactura;
    private int contador_productos;
    private final int MAX_PRODUCTOS = 45;

    //Constructor vacío por buena practica
    public Venta() {

    }

    public Venta(Vendedor vendedor, Cliente cliente, Double total, int[] cantidad, Double[] subTotal, String fecha, int noFactura) {
        this.vendedor = vendedor;
        this.cliente = cliente;
        this.total = total;
        this.cantidad = cantidad;
        this.subTotal = subTotal;
        this.fecha = fecha;
        this.noFactura = noFactura;

        this.contador_productos = 0;
        this.productos_venta = new Producto[MAX_PRODUCTOS];
    }
```

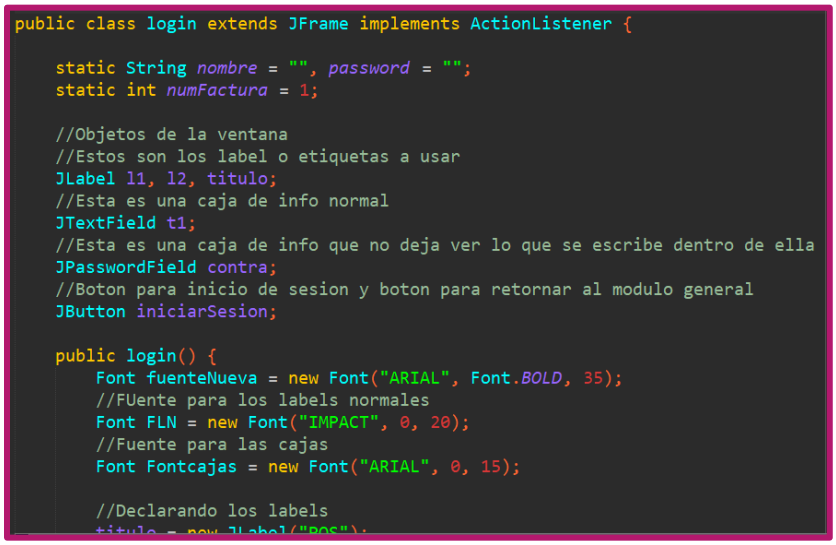
Clase Venta, elaboración propia, 2021

Se puede apreciar en la imagen que dicha clase posee los mismos atributos que mencionamos anteriormente, dicho clase puede almacenar al vendedor, al cliente, y a un arreglo de productos, el cual es el que va ligado a lo máximo de productos que puede llegar a comprar el cliente en una sola venta.

Una vez explicado, la parte de las clases que se utilizaron para poder desarrollar la base del proyecto pasamos a explicar los elementos gráficos del programa.

Login:

Está clase fue desarrollada usando las librerías de swing y awt, para poder crear la interfaz gráfica. En este apartado, se muestra una ventana indicando que ingrese el código y contraseña, para poder ingresar al sistema.

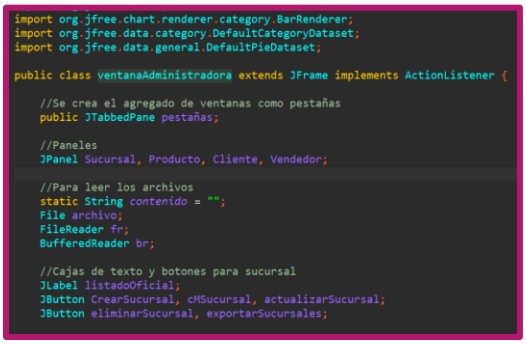


```
public class login extends JFrame implements ActionListener {

    static String nombre = "", password = "";
    static int numFactura = 1;

    //Objetos de la ventana
    //Estos son los label o etiquetas a usar
    JLabel l1, l2, titulo;
    //Esta es una caja de info normal
    JTextField t1;
    //Esta es una caja de info que no deja ver lo que se escribe dentro de ella
    JPasswordField contra;
    //Boton para inicio de sesion y boton para retornar al modulo general
    JButton iniciarSesion;

    public login() {
        Font fuenteNueva = new Font("ARIAL", Font.BOLD, 35);
        //FUente para los labels normales
        Font FLN = new Font("IMPACT", 0, 20);
        //Fuente para las cajas
        Font Fontcajas = new Font("ARIAL", 0, 15);

        //Declarando los labels
        titulo = new JLabel("POS");
```

Clase Login, Elaboración propia, 2021

Como se puede apreciar en la imagen, se muestran algunos componentes de la gráfica en tipo código, el constructor de login, nos permite iniciar esos componentes, para poder mostrarlos en la ventana de login.

Para poder avanzar existen dos caminos en general, uno de ellos es que ingrese el administrador y el otro es que ingrese un vendedor a realizar una venta o varias. En caso de ser el administrador, se redirige a la ventana administradora, la cual es la siguiente:



```
import org.jfree.chart.renderer.category.BarRenderer;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

public class ventanaAdministradora extends JFrame implements ActionListener {

    //Se crea el agregado de ventanas como pestañas
    public JTabbedPane pestañas;

    //Paneles
    JPanel Sucursal, Producto, Cliente, Vendedor;

    //Para leer los archivos
    static String contenido = "";
    File archivo;
    FileReader fr;
    BufferedReader br;

    //Cajas de texto y botones para sucursal
    JLabel listadoOficial;
    JButton CrearSucursal, cMSucursal, actualizarSucursal;
    JButton eliminarSucursal, exportarSucursales;
```

Ventana administradora, Elaboración propia, 2021

Para poder acceder a la ventana de vendedor, la única forma es que el admin ingrese un vendedor nuevo al listado, o que realice una carga masiva de datos de vendedores, cerrando su sesión y volviendo al login, ya podría colocar el nombre del vendedor y la contraseña por defecto que es: “1234”, con eso ya ingresa a la ventana de vendedor, a esa ventana en código su clase se llama ventana de ventas.

```
public class ventasV extends JFrame implements ActionListener, MouseListener {

    //Hay unos datos a eliminar tomarlo en cuenta...
    Producto producto_a_agregar = null;
    Producto[] productos_a_agregar = new Producto[45];
    int[] cantidades = new int[45];
    int contador_cantidades = 0;
    Cliente cliente_compra = null;
    Venta venta_a_realizar;
    Double[] subTotales = new Double[45];
    int cont_productos_agregar = 0;
    int cont_sub_totales = 0;
    String nombre_cliente = "";
    int posicion_cliente = 0;
    int cantidad_a_descontar = 0;
    String fechaActual = LocalDate.now().toString();

    //Se crea el agregado de ventanas como pestañas
    public JTabbedPane pestañas;
```

Ventas Ventana, Elaboración propia, 2021

Hay algunos códigos lógicos de programación, todo esto se mostrará más adelante.

Regresando a la clase de ventana administradora, se puede encontrar 4 pestañas, las cuales, se tendría que visualizar en la ventana, para el código, se encuentran en la misma clase, lo único que cambia es que cada pestaña esta contenida por un propio panel, ósea un panel para sucursales, uno para clientes, etc.

Para las cuatro clases principales se realizan las mismas acciones, exceptuando sucursales, que es la única que no muestra datos en una gráfica. Esto quiere decir que en las cuatro pestañas se pueden realizar las siguientes acciones:

- Crear (Sucursal, Vendedor, Cliente, Producto) de manera manual
- Carga Masiva de datos
- Actualizar (Sucursal, Vendedor, Cliente, Producto) de manera manual
- Eliminar (Sucursal, Vendedor, Cliente, Producto) de manera manual
- Exportar listado (Sucursal, Vendedor, Cliente, Producto)

Para cada uno de ellos se realizó un método, pero al ser repetitivos solo se mostrará el caso de uno solo, lo único en que podrían variar los métodos, es la cantidad de datos que recibe por parámetro el método.

Crear Sucursal:

Cuando en la venta se realiza clic en el botón de crear Sucursal, se manda a llamar el método de agregarSucursales, a la cual se le pasa por parámetro una nueva sucursal de tipo Sucursal, se evalúa el contador de sucursales que hay en existencia, sino se supera la cantidad permitida se agrega, sin problemas. Se adjunta la imagen del código:

```
/*
*Se agregan las sucursales de uno en uno
 */
public static void agregarSucursales(Sucursal sucursal) {
    sucursales[contador_sucursales++] = sucursal;
    escribirSucursales(sucursales);
    ordenarPorCodigoSucursal();
}
```

Agregar Sucursales, Elaboración propia, 2021

```
if (contador_sucursales < sucursales.length) {
    Sucursal nueva_sucursal = new Sucursal(codigo_validado, nombre, direccion, correo, telefono);
    agregarSucursales(nueva_sucursal);
} else {
    JOptionPane.showMessageDialog(this, "Solo se pueden manejar 50 sucursales");
}
```

Validación, Elaboración propia, 2021

Carga masiva Sucursal:

Ahora toca la parte de carga masiva de datos, para lo cual, es necesario usar dos métodos más aparte del de agregar, para este ejemplo usaremos la clase de productos, todas usan el mismo método al empezar, el cual tiene de nombre leerArchivos, este se encarga de leer la información del formato Json y almacenarla en una variable llamada contenido, pero para seleccionar el archivo a trabajar se hizo uso del componente JFileChooser, para esto, se realiza un filtro de los archivos que nos interesa saber, para esto se consigue el nombre de la ruta y se obtiene el nombre del archivo por una línea de código, dependiendo del nombre del archivo, así es como se trabajará.

```java
public void leerArchivos() {
    try {

        JFileChooser fc = new JFileChooser();

        //Creamos el filtro
        FileNameExtensionFilter filtro = new FileNameExtensionFilter("*.JSON", "json");

        //Le indicamos el filtro
        fc.setFileFilter(filtro);

        int op = fc.showOpenDialog(this);
        if (op == JFileChooser.APPROVE_OPTION) {

            archivo = fc.getSelectedFile();

            //Lo que hace es extraer la ruta absoluta del archivo fc.getSelectedFile()
            //Lo volvemos a string usando el metodo toString()
            //Pero lo que se necesita es solamente el name del archivo
            //para ello se usa un indice y veremos que mas, pero funciona
            String fullPath = fc.getSelectedFile().toString();
            int indice = fullPath.lastIndexOf("\\");
            String fileName = fullPath.substring(indice + 1);
```

Leer Archivos, Elaboración propia, 2021

Ahora se adjunta una imagen de las validaciones:

```java
if (fileName.equalsIgnoreCase("Sucursales.json")) {
    //se manda a llamar al metodo convertir a json Sucursales
    contenido_json_a_sucursal();
}

if (fileName.equalsIgnoreCase("Productos.json")) {
    //se manda a llamar al metodo convertir a json Productos
    contenido_json_a_producto();
}

if (fileName.equalsIgnoreCase("Clientes.json")) {
    //se manda a llamar al metodo convertir a json Clientes
    contenido_json_a_cliente();
}

if (fileName.equalsIgnoreCase("Vendedores.json")) {
    //se manda a llamar al metodo convertir a json Vendedores
    contenido_json_a_vendedor();
}
```

Validaciones, Elaboración propia, 2021

En nuestro caso, como se está trabajando con Productos, se manda a llamar el método de contenido Json a producto, el cual se mostrará a continuación:

```java
public static void contenido_json_a_producto() {
    //System.out.println(contenido);

    // JsonParser parser -> Todos los metodos necesarios para interpretar un JSON.
    JsonParser parser = new JsonParser();
    // JsonArray arreglo de objetos JSON.
    JsonArray arreglo = parser.parse(contenido).getAsJsonArray();
    // PARA ESTE MOMENTO, ARREGLO YA TIENE LO QUE ES UN ARREGLO ALMACENADO
    System.out.println("Cantidad de Objetos: " + arreglo.size());

    //Iremos convirtiendo los datos de una
    //vuelta en un objeto de tipo Sucursal
    for (int i = 0; i < arreglo.size(); i++) {

        // JsonObject -> Tomar el Objeto del Arreglo
        JsonObject objeto = arreglo.get(i).getAsJsonObject();

        // GUARDAMOS LOS DATOS EN VARIABLES
        int codigo = objeto.get("codigo").getAsInt();
        String nombre = objeto.get("nombre").getAsString();
        String descripcion = objeto.get("descripcion").getAsString();
        int cantidad = objeto.get("cantidad").getAsInt();
        double precio = objeto.get("precio").getAsDouble();

        if (contador_productos < productos.length) {
            //Se crea un nuevo producto
            Producto nuevoProducto = new Producto(codigo, nombre, descripcion, cantidad, precio);

            //Se manda a llamar al procedimiento de agregar un nuevo producto
            //al arreglo de objetos de tipo producto
            agregarProductos(nuevoProducto);
        } else {
            JOptionPane.showMessageDialog(null, "Solo se pueden manejar mas de 200 productos");
        }
    }
    ordenarPorCodigoProducto();
    contenido = "";
}
```

Contenido Json a producto, Elaboración propia, 2021

Al final se hace la validación y si verdadera, le manda el nuevo producto a agregar producto, se ordenan los productos por medio de su código y se limpia la variable de contenido.

Ahora se mostrará la acción cuando se quiere actualizar un cliente, para eso se utiliza una nueva ventana, donde se admiten los campos a actualizar, en este caso, se admiten los campos, se leen y se cargan a un arreglo que será mandando como parámetro al proceso llamado" actualizarClienteCD", se busca al cliente con tal código y se actualiza la información del cliente.

```java
public static void actualizarClienteCD(String[] cliente) {
    int codigo_a_actualizar = Integer.parseInt(cliente[0]);

    for (int i = 0; i < contador_productos; i++) {
        if (clientes[i].getCodigo() == codigo_a_actualizar) {
            clientes[i].setCodigo(codigo_a_actualizar);
            clientes[i].setNombre(cliente[1]);
            clientes[i].setNit(cliente[2]);
            clientes[i].setCorreo(cliente[3]);
            clientes[i].setGenero(cliente[4]);
            break;
        }
    }
    escribirClientes(clientes);
    contadorGenero();
}
```

Actualizar Cliente, Elaboración propia, 2021

Por último, se mostrará la acción de eliminar, en este caso mostraremos el método de eliminar Vendedor, para lo cual solo se necesita su código. Se buscar y se elimina con el método de eliminar vendedor.

```
int codigo_a_buscar = Integer.parseInt(t1.getText());
if (ae.getSource() == eliminarVendedor) {

    if (eliminarVendedorCD(codigo_a_buscar)) {
        JOptionPane.showMessageDialog(this, "Se elimino al vendedor con el codigo:" + codigo_a_buscar);
    } else {
        JOptionPane.showMessageDialog(this, "No se encontró al vendedor a eliminar con el codigo:" + codigo_a_buscar);
    }
    ventanaAdministradora manejoSucursal = new ventanaAdministradora();
    manejoSucursal.pestañas.setSelectedIndex(3);
    manejoSucursal.setVisible(true);
    dispose();
}
```

Mandando a llamar el método, Elaboración propia, 2021

```
public static boolean eliminarVendedorCD(int codigo_a_buscar) {

    boolean bandera = false;

    //Se busca el codigo, si el codigo no lo encontro debe decir que
    //no existen entre los vendedores
    for (int i = 0; i < contador_clientes; i++) {
        if (vendedores[i].getCodigo() == codigo_a_buscar) {
            bandera = true;
            //Se corren las posiciones hacia la izquierda en el caso de que
            //se encuentre el vendedor a eliminar.
            for (int j = i; j < contador_vendedores; j++) {
                vendedores[j] = vendedores[j + 1];
            }
            contador_vendedores--;
            break;
        }
    }
    mayorCantidadVendedores();
    ordenarPorCodigoVendedor();
    escribirVendedores(vendedores);
    return bandera;
}
```

Eliminar Vendedor, Elaboración propia, 2021

Para ese momento, si se encuentra, ya lo elimino, si no, es porque no lo encontró en el registro de vendedores.

Para las cuatro clases existe un método que ordena su arreglo por medio de su código de manera ascendente. Se tomará solo un método ya que todos realizan el mismo método.

```java
/*
Ordena de menor a mayor a sus objetos segun el codigo del producto
 */
public static void ordenarPorCodigoProducto() {
    //System.out.println("Se ordenan los productos por medio de su codigo");
    int cont = contador_productos;
    for (int i = 0; i < cont - 1; i++) {
        for (int j = 0; j < cont - i - 1; j++) {
            if (productos[j].getCodigo() > productos[j + 1].getCodigo()) {
                Producto aux = productos[j + 1];
                productos[j + 1] = productos[j];
                productos[j] = aux;
            }
        }
    }
}
```

Ordenar por código el arreglo de productos, Elaboración propia, 2021

También en algunos todos es necesaria comprobar que el código que se ingresa no se encuentre repetido, para eso se presenta el método de verificar el producto repetido.

```java
public static boolean verificarProductoNoRep(int codigo_a_buscar) {

    boolean bandera = false;

    //Se busca el codigo, si el codigo no lo encontro debe decir que
    //no se ha agregado a los productos
    for (int i = 0; i < contador_productos; i++) {
        if (productos[i].getCodigo() == codigo_a_buscar) {
            bandera = true;
            break;
        }
    }

    return bandera;
}
```

Verificar producto no repetido, Elaboración propia, 2021

Existe otro método que realizan las cuatro clases principales, las cuales son convertir los datos y retornarlos en un objeto, esto para poder cargar los datos a tabla, que se mostrará respectivamente.

```java
public static Object[][] convertirDatosSucursales() {
    //se trae el contador de sucursales ya que en base a eso se pasaran las filas de datos
    //de los sucursales que se hayan encontrado
    int filas = contador_sucursales;

    Object[][] arregloSucursal = new Object[filas][5];

    for (int i = 0; i < filas; i++) {
        arregloSucursal[i][0] = sucursales[i].getCodigo();
        arregloSucursal[i][1] = sucursales[i].getNombre();
        arregloSucursal[i][2] = sucursales[i].getDireccion();
        arregloSucursal[i][3] = sucursales[i].getCorreo();
        arregloSucursal[i][4] = sucursales[i].getTelefono();
    }
    return arregloSucursal;
}
```

Convertir Datos Sucursales, Elaboración propia, 2021

Luego de eso, esos datos son enviados a una variable llamada datosSucursal, que recibe los datos de las sucursales, listos para cargar a la tabla.

```java
datosSucursal = convertirDatosSucursales();
String[] columnas = {"Codigo", "Nombre", "Direccion", "Correo", "Telefono"};
tablaSucursales = new JTable();

//Se hace uso de la tabla model para validar que el admin no pueda realizar
//ningun tipo de cambio por medio de las tablas!
DefaultTableModel mTS = new DefaultTableModel(datosSucursal, columnas) {

    @Override
    public boolean isCellEditable(int row, int colum) {
        return false;
    }
};
tablaSucursales.setModel(mTS);
```

Cargando los datos a la tabla, Elaboración propia, 2021

Exceptuando Sucursal, el resto necesita de una gráfica para añadir mas información, para ello se utiliza la siguiente parte de código:

```java
DefaultCategoryDataset datasetP = new DefaultCategoryDataset();
datasetP.addValue(cantidades_productos_top[0], String.valueOf(nombres_productos_top[0]), "Producto 1");
datasetP.addValue(cantidades_productos_top[1], String.valueOf(nombres_productos_top[1]), "Producto 2");
datasetP.addValue(cantidades_productos_top[2], String.valueOf(nombres_productos_top[2]), "Producto 3");

chartProducto = ChartFactory.createBarChart3D("Top productos con más existencia", "Categoria", "Valores", datasetP, PlotOrientation.VERTICAL, true, true, false);
chartProducto.setBackgroundPaint(new Color(236, 240, 241));
chartProducto.getTitle().setPaint(new Color(125, 60, 152));
chartProducto.getLegend().setItemPaint(Color.blue);
```

Creación de la gráfica de productos con más existencia, Elaboración propia, 2021

Como mencionamos antes, existía un botón de exportar los datos a un listado, para eso se utilizó un método que reaccione al momento de hacer clic en el botón correspondiente, genera y abre automáticamente el archivo.

Al abrirse el archivo pdf se pueden visualizar los datos de la tabla en la hoja.

```
//Metodo para crear el pdf de crearPDFVendedor
public static void crearPDFVendedor() {
    String nombrePDF = "Listado de Vendedores.pdf";
    try {

        Document documentoVendedores = new Document();
        PdfWriter.getInstance(documentoVendedores, new FileOutputStream(nombrePDF));
        //Se abre el documento para ingresar la tabla
        documentoVendedores.open();
```

Crear PDF Vendedor, Elaboración propia, 2021

Un momento antes de cerrar el método se manda a llamar al proceso de abrir Archivo, al cual lo único que se le pasa por parámetro es el nombre del pdf.

```
        // Iniciar una nueva pagina
        documentoVendedores.newPage();
        //Cerramos el documento para evitar errores
        documentoVendedores.close();
        abrirArchivo(nombrePDF);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Parte final de crear pdf vendedores, Elaboración propia, 2021

Dicha acción se realiza en las 4 principales clases.

Para este momento, en la creación del pdf se usa la librería de itextpdf-5, está librería como lo dice su nombre, nos ayuda a crear un pdf con la información que no interese colocar, la información a mostrar como tales los atributos que se tienen en la clase de vendedores.

Ya que se habló de la ventana del administrador, nos toca viajar a la ventana de los vendedores, en la cual, ellos realizarán varias actividades, entre ellas crear a un nuevo cliente en caso de que no exista en donde lo estamos buscando, utilizar el filtro para encontrarlo, y agregar productos a la tabla.

```
public ventasV(Vendedor vendedor_cargado) {
    vendedor_a_enviar = vendedor_cargado;
    String nombreVendedor = "¡BIENVENIDO " + vendedor_cargado.getNombre() + " !";
    pestañas = new JTabbedPane();

    nuevaVenta = new JPanel();
    nuevaVenta.setLayout(null);

    VentasRealizadas = new JPanel();
    VentasRealizadas.setLayout(null);

    pestañas.addTab("Nueva Venta", nuevaVenta);
    pestañas.addTab("Ventas", VentasRealizadas);

    PestañaNuevaVenta(nuevaVenta);
    PestañaVentas(VentasRealizadas);

    this.setTitle(nombreVendedor);
    this.setSize(950, 675);
    this.setLocationRelativeTo(null);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setResizable(false);
    this.setVisible(true);
```

Clase ventana de Ventas, Elaboración propia, 2021

Se visualiza su constructor al cual se le esta pasando el dato de un vendedor, esto se hizo con el fin de que se pudiera trabajar con cualquier vendedor que hubiera iniciado sesión. Esta ventana contiene algunos métodos, los cuales se mostrarán a continuación.

```
public void cargarNombresClientes() {
    nombreClientes.addItem("");
    ordenarPorCodigoCliente();
    for (int i = 0; i < contador_clientes; i++) {
        nombreClientes.addItem(clientes[i].getNombre());
    }
}
```

Cargar los nombres de los clientes, Elaboración propia, 2021

En este procedimiento, lo que se realiza es colocar los nombres de los clientes en un JComboBox, lo que hace esto es poder guardar dichos datos dentro de una cajita, la cual en el momento de hacer clic sobre ella, muestra los datos en forma de lista.

Para mostrar la primera tabla que es la tabla a la que se le agregan los productos a vender, se utilizo esta parte de código para poder mostrar la tabla:

```
datos = new Object[0][5];
String[] columnas = {"Código", "Nombre", "Cantidad", "Precio", "Subtotal"};
tablaproductos_a_vender = new JTable();

//Se hace uso de la tabla model para validar que el admin no pueda realizar
//ningun tipo de cambio por medio de las tablas!
mTS = new DefaultTableModel(datos, columnas) {

    @Override
    public boolean isCellEditable(int row, int colum) {
        return false;
    }
};
tablaproductos_a_vender.setModel(mTS);
```

Tabla de productos a vender, Elaboración propia, 2021

También se tubo que usar un tipo compuesto, en ese caso hablamos de Map, este únicamente lo usamos, por que acepta valores de “llave”, “valor”, esto se uso debido a que era necesario colocarle diversos atributos a una etiqueta, que, por default, solo permite un atributo a la vez, con dicho método se puede se le agregar el tamaño de letra, el tipo de letra, y otras propiedades como subrayar y colocarle negrita.

```
Font font = filtradosPor.getFont();
Map attributes = font.getAttributes();
attributes.put(TextAttribute.SIZE, new Float(15));
attributes.put(TextAttribute.WEIGHT, TextAttribute.WEIGHT_BOLD);
attributes.put(TextAttribute.UNDERLINE, TextAttribute.UNDERLINE_ON);
filtradosPor.setFont(font.deriveFont(attributes));
```

Generación de varias fuentes en una sola, Elaboración propia, 2021

En dicha ventana contamos con mas de 3 acciones por medio de botones, se describirán 3 de forma inmediata:

- Crear nuevo cliente
- Retornar al login por parte de la pestaña 1
- Retornar el login por parte de la pestaña 2

Al momento de hacer clic en el botón de crear cliente nos llevará a una ventana, la cual nos permitirá llenar los datos de un nuevo cliente. Esta ventana es totalmente diferente a la otra ventana de crear clientes, ya que esta responde únicamente a la ventana de ventas, pero con la diferencia de que el proceso que realiza para agregar al cliente es el mismo de la ventana de cliente de la ventana de administrador.

```
public class nuevoClienteVentas extends JFrame implements ActionListener {

    //Se crean los labels
    JLabel l1, l2, l3, l4, l5, l6, l7;

    //Se crean las cajitas de informacion
    JTextField t1, t2, t3, t4, t5;

    //Se crean los botones
    JButton crearCliente;

    static int codigo_a_buscar = 0;

    Vendedor vendedor_a_enviar;

    public nuevoClienteVentas(Vendedor vendedor_cargado) {
        vendedor_a_enviar = vendedor_cargado;
        Font fuenteNueva = new Font("ARIAL", Font.BOLD, 35);
        Font fuenteNueva2 = new Font("ARIAL", 0, 17);

        l1 = new JLabel("Agregar nuevo cliente");
        l1.setBounds(25, 30, 450, 40);
        l1.setVisible(true);
```

Clase de la ventana de nuevo cliente, Elaboración propia, 2021

Como se puede apreciar en la imagen, esta la clase de nuevo cliente, pero por parte de las ventas, la diferencia radica en que esta clase en su constructor recibe a un vendedor, pero solo es para poder tener continuidad de con quien se está trabajando.

Ahora, se mostrará la parte de código que se menciono anteriormente, que es donde se crea el nuevo cliente y se pasa al método de agregar Clientes.

```
if (contador_clientes < clientes.length) {

    nuevo_cliente = new Cliente(codigo_valido, nombre, nit, correo, genero);
    agregarClientes(nuevo_cliente);
} else {
    JOptionPane.showMessageDialog(this, "Solo se pueden manejar más de 100 clientes");
}
```

Agregar nuevo cliente en la ventana de nuevos clientes, de la ventana de ventas, Elaboración propia, 2021

Nos toca ver como interactuamos con el primer filtro de la ventana de ventas, al momento de realizar clic en el botón de aplicar Filtro, lo que realiza es una búsqueda secuencial de los datos del cliente, ya que se puede buscar por medio de su nombre, nit, correo, genero, y por todos combinados. En dichos campos se valida que si no se ha llenado no se puede realizar ninguna búsqueda, ahora bien, si se llena el campo de nombre o de genero existe la posibilidad de que se repitan, para ese caso se menciona que tendrá que llenar más campos, como el nit o el correo, por ser más explícitos.

```
//valida cuando no se ha llenado ningun valor
if (nombre_a_buscar.equals("") && nit_a_buscar.equals("") && correo_a_buscar.equals("") && genero_a_buscar.equ
    JOptionPane.showMessageDialog(this, "No se lleno ningun campo");
} else {
    //Caso en el que a las 4 cajas se les lleno de informacion
    if (!nombre_a_buscar.equals("") && !nit_a_buscar.equals("") && !correo_a_buscar.equals("") && !genero_a_bu
        pos_correo_encontrado = buscarClientePorCorreo(correo_a_buscar);
        pos_nit_encontrado = buscarClientePorNit(nit_a_buscar);

        cant_genero_encontrado = buscarClientePorGeneroCantRep(genero_a_buscar);
        posGeneroRep = new int[cant_genero_encontrado];
        posGeneroRep = buscarClientePorGeneroCantRepPos(cant_genero_encontrado, genero_a_buscar);
        int posComprobadaGenero = compararDatosGeneroYCorreo(posGeneroRep, pos_correo_encontrado);

        cant_nombre_encontrado = buscarClientePorNombreCantRep(nombre_a_buscar);
        posNombreRep = new int[cant_nombre_encontrado];
        posNombreRep = buscarClientePorNombreCantRepPos(cant_nombre_encontrado, nombre_a_buscar);
        int posComprobadaNombre = compararDatosNombreYNit(posNombreRep, pos_nit_encontrado);

        if (pos_nit_encontrado == pos_correo_encontrado) {

            if (pos_correo_encontrado == posComprobadaGenero) {
```

Filtro de clientes, Elaboración propia, 2021

Cuando se realiza un clic sobre el botón de agregar Producto, se agrega el producto ingresado en la caja de texto, además de eso se tiene que agregar la cantidad que se llevará, una vez llenado ambos campos, ya se puede agregar el producto a la tabla, una vez ingresada la cantidad y el producto se descontará automáticamente la cantidad de productos a ese tipo de producto.

```
(ae.getSource() == agregarProducto) {
 //Validamos que el JComboBox, no este en la posicion vacia
 if (nombreClientes.getSelectedItem().equals("")) {
     JOptionPane.showMessageDialog(this, "Usted no puede agregar ningún producto, "
             + "hasta seleccionar a algún cliente.");
 } else {

     if (coP.equals("") || caP.equals("")) {
         JOptionPane.showMessageDialog(this, "Se deben de llenar ambos campos, codigo y cantidad");
     } else {
         nombre_cliente = nombreClientes.getSelectedItem().toString();
         posicion_cliente = (nombreClientes.getSelectedIndex() - 1);

         //Empezamos la jugada de la tabla
         codigo_producto_a_buscar = Integer.parseInt(codigoProducto.getText());
         cantidad_a_descontar = Integer.parseInt(cantidadProducto.getText());
         producto_a_agregar = buscarCodigoProducto(codigo_producto_a_buscar);//me retorna el producto
         subtotal = (double) (cantidad_a_descontar * producto_a_agregar.getPrecio());
         producto_a_agregar.descontarProductos(cantidad_a_descontar);
         ControlarDatos.escribirProductos(productos);

         //preparando datos para agregarlos a la tabla
         String[] datosProducto = {String.valueOf(producto_a_agregar.getCodigo()),
```

Agregar Producto, elaboración propia, 2021

Al momento de realizar clic en el botón de vender, no se le permitirá vender si la tabla se encuentra vacía, ya que eso significa que no se ha agregado ningún producto a vender, de lo contrario se puede vender. Al momento de hacer la venta, se genera la factura de la venta y se “imprime”, en nuestro caso se abre automáticamente con el programa por default para trabajar con pdf.

```
if (ae.getSource() == vender) {
    int Seleccionada = tablaproductos_a_vender.getRowCount();

    if (Seleccionada != 0) {
        cliente_compra = clientes[posicion_cliente];

        /*
        System.out.println("\n----------------------------------------");
        System.out.println("El cliente a trabajar es: " + nombre_cliente);
        System.out.println("La posicion del cliente es: " + posicion_cliente);
        System.out.println("----------------------------------------\n");
         */
        venta_a_realizar = new Venta(vendedor_a_enviar, cliente_compra, totalVenta,
                cantidades, subTotales, fechaActual, numFactura);

        //Se usara un for para llenar la venta de sus productos
        for (int i = 0; i < cont_productos_agregar; i++) {
            //Se debe agregar el producto a la venta
            //debemos moverlo a donde esta la venta a realizar instanciada, sino no va a jalar
            venta_a_realizar.agregarProducto(productos_a_agregar[i]);
        }
```

Realizar la venta de productos, Elaboración propia, 2021

Cabe mencionar también que cuando se hace la venta, la tabla se limpia y se reinician contadores, con esto me refiero a que se deja limpio todo lo que respecta a los datos de la venta, esto con el fin de que no se acumulen, productos con otra venta y se produzca un error en dicho proceso.

El método de crear PDF venta lo que realiza es el pdf de la venta realizada, y luego de esto se abre automáticamente con el método de abrir Archivo, mencionado anteriormente, la diferencia de este método con el resto de creación de pdf, es que se le envía por parámetro, lo que viene siendo la venta.

```
//aqui mandamos a crear el archivo y a abrirlo automaticamente
crearPDFVenta(venta_a_realizar);
```

Visualización de la llamada al método de crear PDF venta, Elaboración propia

Ahora se mostrará el método en sí, para que se tenga una idea de como funciona.

```
//Metodo para crear el pdf de crearPDFVenta *Terminado*
public static void crearPDFVenta(Venta venta) {
    String nombrePDF = "";
    nombrePDF = "Factura No" + venta.getNoFactura() + ".pdf";
    Producto[] productos_a_imprimir = venta.getProductos_venta();
    int[] cantidades_a_imprimir = venta.getCantidad();
    Double[] subtotales_a_imprimir = venta.getSubTotal();

    int contador_general = 0;
    for (int i = 0; i < productos_a_imprimir.length; i++) {
        if (productos_a_imprimir[i] != null) {
            contador_general++;
        }
    }

    Font fontLugar = new Font(Font.FontFamily.COURIER, 14, Font.BOLD);
    try {
        Paragraph parrafo = new Paragraph();
        Paragraph fecha = new Paragraph();

        Document documentoVenta = new Document();
        PdfWriter.getInstance(documentoVenta, new FileOutputStream(nombrePDF));
```

Crear PDF Venta, Elaboración propia, 2021

La ventana de ventas posee dos pestañas, una es para la parte de las ventas de ese momento, y la otra es un historial de ventas realizadas por el vendedor, cada vendedor tendrá un historial de ventas.

En esta segunda pestaña también se puede filtrar, pero se filtran las ventas, según el numero de factura, el nit, el nombre y la fecha, entre más información se agregue a los campos más específica será la búsqueda. Al momento de dar clic sobre aplicar Filtro, se buscaran las coincidencias y se mostrarán en la tabla.

```
if (ae.getSource() == aplicarFiltroVentas) {
    JButton nuevo = new JButton("Visualizar");
    nuevo.setForeground(Color.BLUE);

    Venta venta_a_mostrar;
    Venta[] ventas_match;
    int coincidencias = 0;

    String no_factura_a_buscar_a = t5.getText();
    int no_factura_a_buscar = 0;
    nit_a_buscar = t6.getText();
    nombre_a_buscar = t7.getText();
    String fecha_a_buscar = t8.getText();
    if (no_factura_a_buscar_a.equals("") && nit_a_buscar.equals("") && nombre_a_buscar.equals("") && fecha_a_buscar.equals("")) {

        JOptionPane.showMessageDialog(this, "Campos vacios!, llenelos", "Advertencia", JOptionPane.WARNING_MESSAGE);

    } else if (!no_factura_a_buscar_a.equals("") && nit_a_buscar.equals("") && nombre_a_buscar.equals("") && fecha_a_buscar.equals("")) {
        no_factura_a_buscar = Integer.parseInt(no_factura_a_buscar_a);
        //Buscamos unicamente con el numero de factura
        venta_a_mostrar = venta_a_pdf_fac(vendedor_a_enviar.getVentas(), no_factura_a_buscar);

        if (venta_a_mostrar != null) {
            reset_mTRV();
            nuevo.setName(String.valueOf(venta_a_mostrar.getNoFactura()));
            Object[] datosCoincidencia = {
                venta_a_mostrar.getNoFactura(),
                venta_a_mostrar.getCliente().getNit(),
                venta_a_mostrar.getCliente().getNombre(),
```

Filtrar ventas, Elaboración propia, 2021

Cuando se realiza la búsqueda, la tabla se limpia y se insertan los datos coincidentes con la búsqueda.

Por último dentro de la tabla se puede realizar una acción, y es que se puede imprimir nuevamente la factura, para comprobar los productos vendidos, el nombre y nit del cliente, la caja donde se le atendió y el nombre del vendedor.

Para serializar se utilizaron 3 métodos:

- Leer + (Tipo de objeto) (se pasa por parámetro el arreglo)
- Cargar + (Tipo de objeto)
- Serializar + (Tipo de objeto)

Las 5 clases más un dato primitivo lo utilizan, esto es para poder recuperar los datos y seguir trabajando de manera normal, con los datos ya cargados.

Para tomar ejemplo, usaremos la clase de sucursales.

Para poder serializar el objeto o los objetos es necesario usar una interface que nos ayude a serializar. En este caso se llama Serializable.

```
import java.io.Serializable;

public class Sucursal implements Serializable
```

Ejemplo clase Serializada, Elaboración propia, 2021

Para el proceso de escribir Sucursales se tiene que usar un objeto tipo ObjectOutputStream, que es el que controla el flujo de objetos y también lo que es el FileOutputStream que controla el flujo de archivos, para esto simplemente se debe de crear un archivo "**.bin**"**,** en ese objeto se estarán escribiendo las sucursales, y por ultimo se cierra el objeto para no tener problemas.

```
public static void escribirSucursales(Object sucursales) {
    try {
        objSalida = new ObjectOutputStream(new FileOutputStream("Sucursales.bin"));

        objSalida.writeObject(sucursales);

        objSalida.close();

        //System.out.println("Objeto Serializado");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Procedimiento escribir Sucursales, Elaboración propia, 2021

Para el proceso de cargar Datos Sucursales se tiene que crear un objeto de tipo Object y ahora necesitamos un ObjectInputStream, y un FileInputStream, el primero es para controlar el flujo de entrada de objetos y el segundo es para controlar el flujo de archivos de entrada, hay que tomar en cuenta que el nombre que se usa para escribir los archivos debe ser igual, y se tiene que leer con el método de readObject, en caso de que lo encontremos retornará el objeto, sino retornará null.

```
public static Object cargarDatosSucursales() {
    Object object;
    try {
        ois = new ObjectInputStream(new FileInputStream("Sucursales.bin"));
        object = ois.readObject();
        return object;
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }

    return null;
}
```

Cargar Datos Sucursales, Elaboración propia, 2021

El tercer método nos servirá para poder cargar los datos al arreglo, se tiene que realizar un casteo explicito, para poder pasar los datos del objeto al arreglo, y una vez hecho eso, si el método anterior no devuelve null, se tendría que hacer un conteo de cuantas posiciones del arreglo de sucursales son diferente de null osea de vacío, que no tienen nada, esto se hace para poder llevar la cuenta de que espacios están llenos y así no recorrer todo el arreglo en cada momento.

```
public static void serializarSucursal() {
    sucursales = (Sucursal[]) cargarDatosSucursales();
    /*
    Al momento de que se cargan los datos en el arreglo de sucursales, se tiene que
    hacer nuevamente un conteo de cuantas sucursales hay, esto es para que se le
    pueda asignar al contador_sucursales, y  seguir llevando la cuenta actualizada,
    luego se tiene que cargar estos datos a la tabla.
     */
    if (sucursales == null) {
        System.out.println("No se ha creado el archivo");
        sucursales = new Sucursal[50];
    } else if (sucursales != null) {
        //se va a buscar cuantos datos del arreglo de sucursales es diferente de nulo
        int contador_sucursales_cargadas = 0;
        for (int i = 0; i < sucursales.length; i++) {
            if (sucursales[i] != null) {
                contador_sucursales_cargadas++;
            } else if (sucursales[i] == null) {
                //En el momento en que la sucursal tenga datos vacios,
                //es mejor paralo con un break, para no gastar mas recursos
                break;
            }
        }
        contador_sucursales = contador_sucursales_cargadas;
        //System.out.println("El contador de sucursales tiene: " + contador_sucursales+" sucursales");
        //mostrarSucursales();
    }
}
```

Serialización Sucursal, Elaboración propia, 2021